ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Herramientas para Big Data: Entorno Hadoop.

AUTOR: Francisco Miguel Rodríguez Sánchez
DIRECTOR: Diego Alonso Cáceres
Septiembre / 2014

Índice general

Índice de figuras

Capítulo 1. Introducción.

Actualmente vivimos en la era de los datos. La explosión de datos está sucediendo a todos los niveles en todos los dispositivos electrónicos, aplicaciones, individuos y organizaciones. Cada día se genera una cantidad ingente de datos que hace necesario medirlos a escala de cientos de *petabytes* y, por tanto, ser capaces de administrar y analizar esta gran cantidad de información se convierte en un requisito indispensable dentro del ámbito empresarial. Además el desarrollo de técnicas y tecnologías para el análisis de datos favorece, sin duda, a un mejor conocimiento de la sociedad y de sus necesidades.

El término *Big Data* se refiere a la tendencia en el avance de la tecnología que ha abierto las puertas hacia un nuevo enfoque de entendimiento y toma de decisiones, la cual es utilizada para describir enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos a un base de datos convencional para su análisis. De tal manera que, el concepto de Big Data aplica para toda aquella información que no puede ser procesada o analizada utilizando procesos o herramientas tradicionales. Sin embargo, Big Data no se refiere a alguna cantidad en específico, ya que es usualmente utilizado cuando se habla en términos de petabytes o exabytes de datos. Lo que para una determinada empresa es Big Data puede podría no serlo para otra, esto depende de los recursos de los que cada una disponga.

Para poder extraer valor de Big Data, se necesita un modo alternativo de procesar los datos. Una de las principales herramientas capaces de tratar e indagar en estos datos para extraer información de ellos se denomina Hadoop. Se trata de una plataforma de código libre y desarrollada por Apache que permite el procesamiento de grandes volúmenes de datos a través de clústeres, usando un modelo simple de programación. Además, su diseño permite pasar de pocos nodos a miles de nodos de forma ágil. Básicamente, Hadoop proporciona dos cosas: un sistema de archivos distribuidos (HDFS, Hadoop File System) y un framework de MapReduce (paradigma de programación) que permite dividir y paralelizar los cálculos entre un número indefinido de ordenadores de bajo coste. Del uso de un clúster de computadores convencionales en el análisis de datos radica una de las mayores ventajas de Apache Hadoop.

El presente trabajo fin de grado tiene como objetivo presentar y estudiar la herramienta Apache Hadoop para el procesamiento de Big Data. Para ello, se realizaran una serie de ejemplos además del análisis de archivos provenientes del tráfico generado en la Teleco LAN Party 6 (TLP6) durante el primer día de celebración. El propósito del trabajo es evaluar la tecnología y todas las herramientas que se derivan de ella con la intención de mostrar todas sus posibilidades en el tratamiento de cualquier tipo de datos. Además se abordará la instalación y configuración paso a paso de la herramienta para distintas versiones.

Apache Hadoop es actualmente la herramienta libre más utilizada para el análisis de Big Data, si bien también existen alternativas diseñadas para problemas específicos y, por tanto, puede ser más acertado utilizar otro tipo de herramienta para el análisis de datos:

- R: es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se trata de un proyecto de software libre resultado de la implementación GNU del premiado lenguaje S.

- SAS: se trata de un paquete que comprende un conjunto de programas de análisis estadístico de datos. Con esta herramienta se pueden realizar diferentes tipos de trabajo como almacenar o recuperar información, modificar la información existente y realizar estadística y análisis complejos de los datos.

- SPSS: es un programa estadístico informático muy usado en las ciencias sociales y las empresas de investigación de mercado. Esta desarrollado en JAVA y es propiedad de IBM.

- MONGODB: es un sistema de base de datos NoSQL (no relacional) orientado a documentos y desarrollado bajo el concepto de código abierto.

- DISCO PROJECT: se trata de un framework de código abierto para computación distribuida basado en el paradigma de programación MapReduce.

- STORM: es un sistema de computación distribuida a tiempo real de código abierto. Storm hace que sea sencillo procesar de manera fiable flujos de datos.

En el siguiente capítulo, el estado de la técnica, se describirán con más detalle las alternativas propuestas, se explicarán las opciones de cada una de ellas, sus inconvenientes y se verá cómo muchas de ellas son combinables con Hadoop. Además se aclararán los motivos de la elección de Apache Hadoop como la herramienta a presentar durante el trabajo fin de grado.

En el tercer capítulo, se abordará la explicación detallada de la arquitectura Apache Hadoop y porqué es útil dentro de la sociedad actual. Se explicarán todas sus funcionalidades y aspectos relevantes así como sus posibles opciones de instalación y configuración (Ver *Anexo A*) para comenzar con el análisis de los primeros ficheros.

Se dejará para el cuarto capítulo el análisis de los archivos de provenientes del tráfico generado durante la TLP6. Para su tratamiento se utilizará un lenguaje de programación creado para Apache Hadoop: Pig. Se trata de un lenguaje de alto nivel desarrollado por Apache cuyo objetivo es la creación de programas de análisis de datos.

Además de todos los capítulos dedicados al análisis de la plataforma, en el capítulo 5 se plantearán posibles ampliaciones y trabajos futuros que puedan continuar el trabajo realizado. Finalmente, se explicarán las conclusiones personales derivadas de este trabajo fin de grado.

Capítulo 2. Estado de la técnica.

Como se citó en la introducción, existen alternativas para el análisis de datos a Hadoop. En este capítulo se van explicar todas aquellas plataformas que se enunciaron durante el primer capítulo.

Como ya se ha mencionado, Hadoop proporciona dos elementos principales: un framework MapReduce y un sistema de archivos distribuido (HDFS). Por una parte, el paradigma de programación MapReduce permite crear algoritmos de análisis de datos para extraer valor de ellos a través del estudio de los resultados. Por otra parte, el sistema de archivos (HDFS) provee de una base de datos tolerante a fallos y con una alta disponibilidad. Por tanto, Hadoop proporciona una doble funcionalidad que en algunos casos las plataformas alternativas no van a poder ofrecer. Es decir, alguna de las herramientas cumplen únicamente una de las funcionalidades de Hadoop aunque de un modo muy eficiente. Varias de las herramientas pueden utilizarse dentro del entorno Hadoop para poder completar esta carencia.

Las plataformas que se van a desarrollar en este apartado pueden ser divididas en tres grandes bloques:

- Herramientas para el análisis de datos: No proveen de un sistema de gestión de archivos pero tienen gran capacidad para el análisis de ficheros de datos. Sobre este grupo se van a comentar tres plataformas: R, SAS, SPSS.

- Bases de datos no relacionales: Proveen de un sistema de gestión de archivos distribuido capaz de manejar enormes cantidades de datos. Por contra, es necesario utilizar una herramienta complementaria para analizar los ficheros de datos. Dentro de este grupo se explicará la base de datos MongoDB.

- Proyectos similares a Hadoop: Debido al éxito de Hadoop están surgiendo herramientas que intentan conseguir sus dos principales funcionalidades utilizando el mismo o distinto paradigma de programación. Estas herramientas se encuentran aún en sus primeras versiones y algunas no son demasiado utilizadas. Dentro de este grupo se desarrollaran los proyectos Disco project y Storm.

2.1. R

R es un lenguaje de programación especialmente indicado para el análisis estadístico.

Figura 1: Logo R

R, cuyo logotipo se muestra en la figura 1, es un lenguaje Orientado a Objetos, y bajo este complejo término se esconde la simplicidad y flexibilidad de R. El hecho que R sea un lenguaje de programación puede desaminar a muchos usuarios que no tienen aptitudes en este tipo de tecnología. Esto no es necesariamente un inconveniente por dos razones. Primero R es un lenguaje interpretado (por ejemplo, Java) y no compilado (como C, C++, Fortran o Pascal) lo cual implica que los comandos escritos en el teclado son ejecutados directamente sin necesidad de construir ejecutables. Además la sintaxis de R es muy simple e intuitiva. Por ejemplo, una regresión lineal se puede ejecutar con el comando lm(y~x). Para que una función sea ejecutada en R debe estar siempre acompañada de paréntesis, inclusive en el caso que no haya nada dentro de los mismos (por ejemplo, ls()). Si se escribe el nombre de la función sin los paréntesis, R mostrará el contenido (código) mismo de la función.

Orientado a Objetos significa que las variables, datos, funciones o resultados se guardan en la memoria activa del computador en forma de objetos con un nombre específico. El usuario puede modificar o manipular estos objetos con operadores (aritméticos, lógicos, y comparativos) y funciones (que a su vez son objetos).

El uso y funcionamiento de los operadores es relativamente intuitivo. Una función en R se puede delinear del siguiente modo esquematizado en la figura 2:

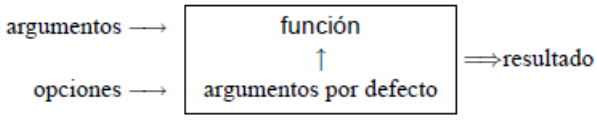


Figura 2: Esquema función R

Los argumentos pueden ser objetos (“datos”, formulas o expresiones), algunos de los cuales pueden ser definidos por defecto en la función. Una función en R puede carecer totalmente de argumentos, ya sea porque todos están definidos por defecto o porque la función realmente no tiene argumentos.

R fue inicialmente diseñado por Robert Gentleman y Ross Ihaka, miembros del Departamento de Estadística de la Universidad de Auckland, en Nueva Zelanda. No obstante, una de las grandes ventajas de R es que hoy en día es, en realidad, fruto del esfuerzo de miles de personas en todo el mundo que colaboran en su desarrollo.
Por otra parte, R se considera la versión libre de otro programa propietario, llamado S (creado por John Chambers) o S-Plus, desarrollado por los Laboratorios Bell. Aunque las diferencias entre R y S son importantes, la mayoría del código escrito para S funciona en R sin modificaciones.

El código de R está disponible como software libre bajo las condiciones de la licencia GNU-GPL, y puede ser instalado tanto en sistemas operativos tipo Windows como en Linux o MacOSX. Desde la página principal se puede acceder tanto a los archivos necesarios para su instalación como al resto de recursos del proyecto R [Ref1, WWW].

Principales objetos R para el análisis de datos.

En el lenguaje de R, los elementos u objetos que se vayan definiendo, bien por el propio usuario, bien como resultado de funciones, deben ser distinguidos para un uso correcto. Por ejemplo, una matriz, por su propia definición, es una colección de números configurados en filas y columnas, todas ellas de la misma longitud. Sin embargo, en ocasiones es necesario reunir números en vectores y estos en algún objeto cuando no todos ellos tienen la misma dimensión, esto es posible en un tipo especial de objeto llamado lista. Desde luego, una lista no es una matriz, luego, aunque sirva para introducir en ella vectores de dimensiones distintas, no admite las operaciones matriciales habituales. No se pretende ser exhaustivo en la descripción de los tipos de objeto de R y, por tanto, solo se van a enunciar los más utilizados para el análisis de datos:

- Vectores.
- Matrices.
- Hojas de datos (Data frames)

Vectores.

Un vector en R puede contener una colección de números o de caracteres no numéricos. Para definir un vector, por ejemplo, el vector x = (1, 3, 5), se usaría la orden: `x<-c(1,3,5)` como se ha realizado en la figura 3.

Notar que se utiliza el operador asignación <- aunque el operador = es igualmente valido (en las últimas versiones de R). Reseñar también que es la función de concatenación c() la que construye el vector.

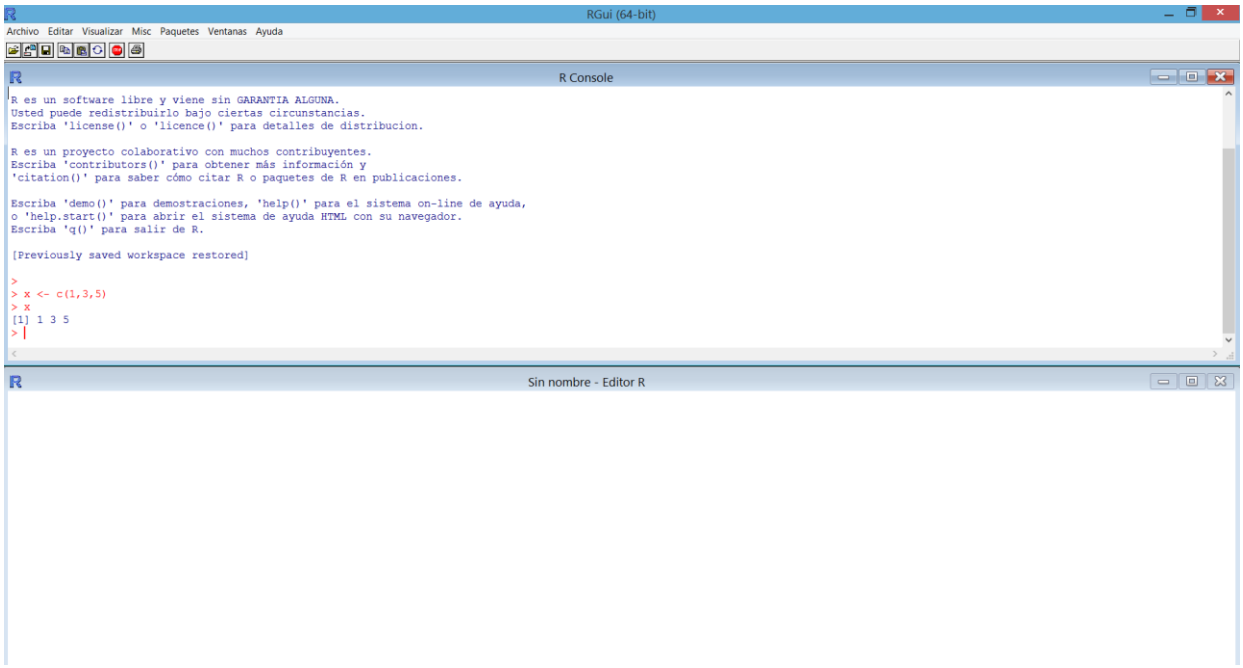


Figura 3:Consola y editor de R

De forma más general, la función seq() permite definir secuencias desde un inicio hasta un fin con una determinada separación entre ellos. Por ejemplo, y<-seq(-3,3,0.5) proporciona:

[1] -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0

También es útil la función rep() para definir vectores como repetición de otros vectores. Por ejemplo, rep(0,100) devolvería un vector de 100 ceros. O también, rep(1:3,3) devolvería:

[1] 1 2 3 1 2 3 1 2 3

Si se desea saber la longitud de un vector, se usará length(). Por ejemplo, length(y) devolvería el valor 13. Comentar, por último, que no hay problema en que un vector en lugar de incluir números incluya caracteres, siempre que éstos estén entre comillas. Por ejemplo, se podría definir el vector `genero=c("Mujer","Hombre")`.

***Matrices*.**

Una matriz se define mediante la función matrix() a la que hay que especificar sus elementos y su dimensión. Por ejemplo, para definir la matriz

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

se utilizaría:

```
matriz<-matrix(c(1,2,3,4,5,6,7,8,9),3,3)
```

Las dimensiones (nº de filas y columnas) de la matriz pueden obtenerse mediante la función dim(). Por ejemplo, dim(matriz) proporcionaría el valor 3 3. Si se desea llegar a elementos concretos de una matriz se hará utilizando corchetes para indicar filas y columnas. Para realizar una multiplicación de dos matrices se utiliza el operador %*%.

***Data frames u hojas de datos*.**

Las hojas de datos constituyen la manera más eficiente en que R puede analizar un conjunto estadístico de datos. Habitualmente se configuran de tal manera que cada fila se refiere a un elemento de la muestra que se analiza, mientras que cada columna hace referencia a las distintas variables analizadas. Con una nomenclatura estadística, diríamos que las filas son los casos y las columnas son las variables. Esa configuración hace que visualmente una hoja de datos parezca una matriz. Sin embargo, como objetos de R, son cosas distintas. Se va a explicar cómo se construye una hoja de datos con los datos de 3 personas, que incluye el color de sus ojos como factor, su peso y su altura. Se comenzaría definiendo el color de los ojos:

```
ojos=factor(c("Azules","Marrones","Marrones"),
levels=c("Azules","Marrones","Verdes","Negros"))
```

Supongamos que los pesos y las alturas son, respectivamente, 68, 75, 88 y 1.65, 1.79, 1.85. Entonces, se definiría la hoja de datos mediante:

```
datos=data.frame(Color.ojos=ojos,Peso=c(68,75,88),Altura=c(1.65,1.79,1
.85))
```

Así, se tendrán tres variables, llamadas Color.ojos, Peso y Altura.

Es posible forzar a que una matriz se convierta en una hoja de datos mediante la función as.matrix. Por ejemplo, `datos2<-as.data.frame(matriz)` convertiría matriz en una hoja de datos. Si se utiliza names(datos2) se verán los nombres que para las variables ha elegido por defecto R:

[1] "V1" "V2" "V3"

Si se desea modificar esos nombres de las variables, se utiliza de nuevo la función names(), forzando la asignación:

```
names(datos2)<-c("Variable 1","Variable 2","Variable 3")
```

El modo en que se puede acceder a los elementos de una hoja de datos es doble:

1. Se puede usar el mismo método que para las matrices.
2. Se puede utilizar el operador $, de la siguiente manera. Para obtener los datos de la variable Color.ojos, por ejemplo, se escribiría datos$Color.ojos.

Para saber el número de filas y de columnas de una hoja de datos se pueden utilizar las funciones nrow() y ncol(). Por ejemplo, ncol(datos) es 3.

Funciones más utilizadas.

A continuación se van a indicar las principales funciones de R. Se pueden encontrar todas las funciones matemáticas simples (log, exp, log10, log2, sin, cos, tan, asin, acos, atan, abs, sqrt. . .), funciones especiales (gamma, digamma, beta, bessell. . .), así como diversas funciones útiles en estadística. Algunas de estas funciones se detallan en la siguiente figura 4.

`sum(x)`	suma de los elementos de x
`prod(x)`	producto de los elementos de x
`max(x)`	valor máximo en el objeto x
`min(x)`	valor mínimo en el objeto x
`which.max(x)`	devuelve el índice del elemento máximo de x
`which.min(x)`	devuelve el índice del elemento mínimo de x
`range(x)`	rango de x o `c(min(x), max(x))`
`length(x)`	número de elementos en x
`mean(x)`	promedio de los elementos de x
`median(x)`	mediana de los elementos de x
`var(x)` o `cov(x)`	varianza de los elementos de x (calculada en $n-1$); si x es una matriz o un marco de datos, se calcula la matriz de varianza-covarianza
`cor(x)`	matriz de correlación de x si es una matriz o un marco de datos (1 si x es un vector)
`var(x, y)` o `cov(x, y)`	covarianza entre x y y, o entre las columnas de x y y si son matrices o marcos de datos
`cor(x, y)`	correlación lineal entre x y y, o la matriz de correlación si x y y son matrices o marcos de datos

Figura 4: Principales funciones matemáticas en R.

Estas funciones devuelven un solo valor (o un vector de longitud 1), a excepción de range() que retorna un vector de longitud 2, y var(), cov(), y cor() que pueden devolver matrices. Las siguientes funciones, mostradas en la figura 5, pueden devolver vectores más complejos:

`round(x, n)`	redondea los elementos de `x` a `n` cifras decimales
`rev(x)`	invierte el orden de los elementos en `x`
`sort(x)`	ordena los elementos de `x` en orden ascendente; para hacerlo en orden descendente: `rev(sort(x))`
`rank(x)`	alinea los elementos de `x`
`log(x, base)`	calcula el logaritmo de `x` en base `"base"`
`scale(x)`	si `x` es una matriz, centra y reduce los datos; si se desea centrar solamente utilizar `scale=FALSE`, para reducir solamente usar `center=FALSE` (por defecto `center=TRUE, scale=TRUE`)
`pmin(x,y,...)`	un vector en el que el *i*avo elemento es el mínimo de `x[i], y[i], ...`
`pmax(x,y,...)`	igual que el anterior pero para el máximo
`cumsum(x)`	un vector en el que el *i*avo elemento es la suma desde `x[1]` a `x[i]`
`cumprod(x)`	igual que el anterior pero para el producto
`cummin(x)`	igual que el anterior pero para el mínimo
`cummax(x)`	igual que el anterior pero para el máximo
`match(x, y)`	devuelve un vector de la misma longitud que `x` con los elementos de `x` que están en `y` (`NA` si no)
`which(x == a)`	devuelve un vector de los índices de `x` si la operación es (`TRUE`) (en este ejemplo, los valores de `i` para los cuales `x[i] == a`). El argumento de esta función debe ser una variable de tipo lógico
`choose(n, k)`	calcula el número de combinaciones de k eventos en n repeticiones = $n!/[(n-k)!k!]$
`na.omit(x)`	elimina las observaciones con datos ausentes (`NA`) (elimina la fila correspondiente si `x` es una matriz o un marco de datos)
`na.fail(x)`	devuelve un mensaje de error si `x` contiene por lo menos un `NA`
`unique(x)`	si `x` es un vector o un marco de datos, devuelve un objeto similar pero suprimiendo elementos duplicados
`table(x)`	devuelve una tabla con el número de diferentes valores de `x` (típicamente para enteros o factores)
`subset(x, ...)`	devuelve una selección de `x` con respecto al criterio (`...`, típicamente comparaciones: `x$V1 <10`); si `x` es un marco de datos, la opción `select` proporciona las variables que se mantienen (o se ignoran con `-`)
`sample(x, size)`	remuestrea al azar y sin reemplazo `size` elementos en el vector `x`; la opción `replace = TRUE` permite remuestrear con reemplazo

Figura 5: Funciones más complejas en R.

Aparte de las funciones que facilita R, se pueden crear funciones propias lo cual será fundamental para el análisis de datos. Para ello se dispone del objeto *function*. La sintaxis es sencilla y se asemeja mucho a otros lenguajes de programación:

```
NombreFuncion <- function (Argumento1,..., ArgumentoN)
{
 expresión 1
 ...
 expresión N
 }
```

Si se desea obtener ayuda sobre el uso de alguna función cuyo nombre es conocido, se puede utilizar la ayuda de R simplemente antecediendo el nombre de esa función con un signo de interrogación. Por ejemplo, ?sort abrirá una ventana del explorador con todos los detalles sobre el uso de esa función, incluyendo ejemplos.

Importar/Exportar datos para su análisis.

Para poder analizar datos, es necesario importarlos o cargarlos al entorno de trabajo. Para ello se va a explicar cómo utilizar las funciones necesarias para importar/exportar datos en R.

R puede leer datos guardados como archivos de texto mediante la función read.table (con sus parámetros):

```
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".",
row.names, col.names, as.is = FALSE, na.strings = "NA", colClasses =
NA, nrows = -1, skip = 0, check.names = TRUE, fill =
!blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#")
```

La función read.table crea un data frame (hoja de datos) y constituye la manera más usual de analizar los datos en R. En la figura 6 se muestran sus principales parámetros.

file	el nombre del archivo (entre "" o como una variable de tipo caracter), posiblemente con su dirección si se encuentra en un directorio diferente al de trabajo (el símbolo \no es permitido y debe reemplazarse con /, inclusive en Windows), o una dirección remota al archivo tipo URL (http://...)
header	una variable lógica (FALSE (falso) o TRUE (verdadero)) indicando si el archivo contiene el nombre de las variables en la primera fila o línea
sep	el separador de campo usado en el archivo; por ejemplo sep="\t" si es una tabulación
quote	los caracteres usados para citar las variables en modo caracter
dec	el caracter usado para representar el punto decimal
row.names	un vector con los nombres de las líneas de tipo caracter o numérico (por defecto: 1, 2, 3, ...)
col.names	un vector con los nombres de las variables (por defecto: V1, V2, V3, ...)
as.is	controla la conversión de variables tipo caracter a factores (si es FALSE) o las mantiene como caracteres (TRUE); as.is puede ser un vector lógico o numérico que especifique las variables que se deben mantener como caracteres
na.strings	el valor con el que se codifican datos ausentes (convertido a NA)
colClasses	un vector de caracteres que proporciona clases para las columnas
nrows	el número máximo de líneas a leer (se ignoran valores negativos)
skip	el número de líneas ignoradas antes de leer los datos
check.names	si es TRUE, chequea que el nombre de las variables sea válido para R
fill	si es TRUE y todas las filas no tienen el mismo número de variables, agrega "blancos"
strip.white	(condicional a sep) si es TRUE, borra espacios extra antes y despues de variables tipo caracter
blank.lines.skip	si es TRUE, ignora líneas en "blanco"
comment.char	un caracter que define comentarios en el archivo de datos; líneas que comienzen con este caracter son ignoradas en la lectura (para desactivar este argumento utilize comment.char ="")

Figura 6: Parámetros de la función read.table en R.

La función write.table guarda el contenido de un objeto en un archivo. El objeto es típicamente un data frame pero puede ser cualquier otro tipo de objeto (vector o matriz). Los argumentos (ver Figura 7) y opciones configurables son los siguientes:

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol
= "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
qmethod = c("escape", "double"))
```

x	el nombre del objeto a exportar
file	el nombre del archivo (por defecto, el objeto se muestra en la pantalla)
append	si es TRUE anexa los datos al archivo sin borrar datos ya existentes en el mismo
quote	lógico o numérico : si es TRUE variables de tipo caracter y factores se escriben entre ; si es un vector numérico, este indica el número de las variables a ser es mostradas entre (en ambos casos los nombres de las variables se escriben entre pero no si quote = FALSE)
sep	el separador de campo utilizado en el archivo
eol	el caracter que indica el final de línea ("\n" es 'retorno')
na	el caracter a usarse para datos faltantes
dec	el caracter usado para el punto decimal
row.names	una opción lógica que indica si los nombres de las líneas se escriben en el archivo
col.names	identificación para los nombres de las columnas
qmethod	si es quote=TRUE, especifica la manera como se debe tratar las comillas dobles "en variables tipo caracter: si es "escape"(o "e", por defecto) cada "es reemplazada por \; si es "dçada "es reemplazada por

Figura 7: Parámetros de la función write.table en R.

R & Hadoop.

Como se mencionó en la introducción, algunas de las herramientas que se proponían como alternativas, eran combinables con Hadoop. Este es el caso de R, que al ser un software libre la comunidad de desarrolladores ha creado un paquete que permite ejecutar en R en un clúster Hadoop. Este paquete se conoce como Rhadoop, se trata de una colección de cuatro paquetes que permiten a los usuarios manejar y analizar datos mediante R en un cluster Hadoop. Estos paquetes han sido implementados y testeados por la empresa Cloudera y están disponibles en sus versiones pre instaladas de Hadoop CDH3 y CDH4 [Ref2, WWW].

2.2. SAS

SAS (Statical Analysis System), cuyo logotipo se muestra en la figura 8, es un conjunto de software estadístico desarrollado por el SAS institute para el análisis avanzado de datos, business intelligence (inteligencia de negocio), manejo de datos y análisis predictivos. Es uno de los softwares más utilizados en el mundo empresarial para el análisis de datos [Ref3, WWW].

Figura 8: Logo de SAS

SAS está compuesto de módulos que tienen tareas específicas, los cuales son vendidos separadamente por el instituto SAS. Entre los módulos se encuentra el BASICO (BASE), que permite la manipulación de datos y obtener algunas tablas y medidas básicas, el STAT, que permite realizar los más variados análisis estadísticos, el IML, el cual es un lenguaje matricial muy similar a GAUSS o a MatLab. Entre otros módulos existe el OR, para investigación de operaciones, el QC, para control de calidad, el ETS, para el análisis de series de tiempo, y muchos otros. Inclusive el Instituto SAS ofrece un compilador propio de C++ para el desarrollo de programas que puedan integrarse al sistema SAS creado por el propio usuario. En este apartado únicamente se explicarán tareas realizadas con el modulo BASE ya que se trata del componente SAS mas utilizado para el análisis de datos.

El funcionamiento de SAS se centra alrededor de tres tareas dirigidas a datos almacenados denominados datasets (conjuntos de datos):

- Acceso a datos, crear y acceder a los datos que se requieren (ficheros, bases de datos, etc.).
- Manejo de datos, da a los datos el formato que la aplicación requiere.
- Análisis de datos, reduce o transforma los datos planos en información útil y significativa.

SAS fue concebido por Anthony J. Barr en 1966. Como estudiante de la Universidad Estatal de Carolina del Norte de posgrado 1962-1964, Barr había creado un análisis del lenguaje de modelado variación inspirada en la notación estadística de Maurice Kendall, seguido de un programa de regresión múltiple que genera el código máquina para realizar transformaciones algebraicas de los datos brutos. Sobre la base de esos programas y su experiencia con los archivos de datos estructurados, creó SAS, introduciendo procedimientos estadísticos en un marco archivo con formato. De 1966 a 1968, Barr desarrolló la estructura y el lenguaje de SAS fundamental. En 1971, SAS fue ganando popularidad dentro de la comunidad académica. Una de las ventajas del sistema se analiza experimentos con los datos que faltan, que era útil para la industria farmacéutica y agrícola, entre otros. En 1973, John Sall se unió al proyecto, haciendo importantes contribuciones de programación en la econometría, series de tiempo, y el

álgebra matricial. En 1976, SAS Institute, Inc. fue incorporado por Barr, Goodnight, Sall, y Helwig. Como hito resaltable, en 2010 se introdujo una versión gratuita de SAS para estudiantes.

Estructura general de un programa SAS.

En la Figura 9 se muestra la estructura que posee un programa SAS. El paso de datos (DATA step) toma los datos y crea una estructura denominada data set (conjunto de datos) en un formato especial que contiene los datos y sus características, los cuales serán después utilizados para los respectivos análisis.

Un programa SAS consiste de instrucciones que terminan con punto y coma (;). Una instrucción es un conjunto de palabras reservadas, nombres o identificadores y operadores. Las instrucciones en un programa SAS se escriben en formato libre, pueden empezar y finalizar en cualquier parte de una línea. Algunas de estas instrucciones son agrupadas en "steps" o pasos y las demás se denominan instrucciones globales. Hay dos tipos de steps:

- DATA STEP o paso de datos que permiten la creación de conjuntos de datos SAS (SAS data sets), estos comienzan con la palabra reservada DATA.
- PROC STEP o paso de procedimientos, ejecutan un proceso o procedimiento SAS, comienzan con la palabra reservada PROC.

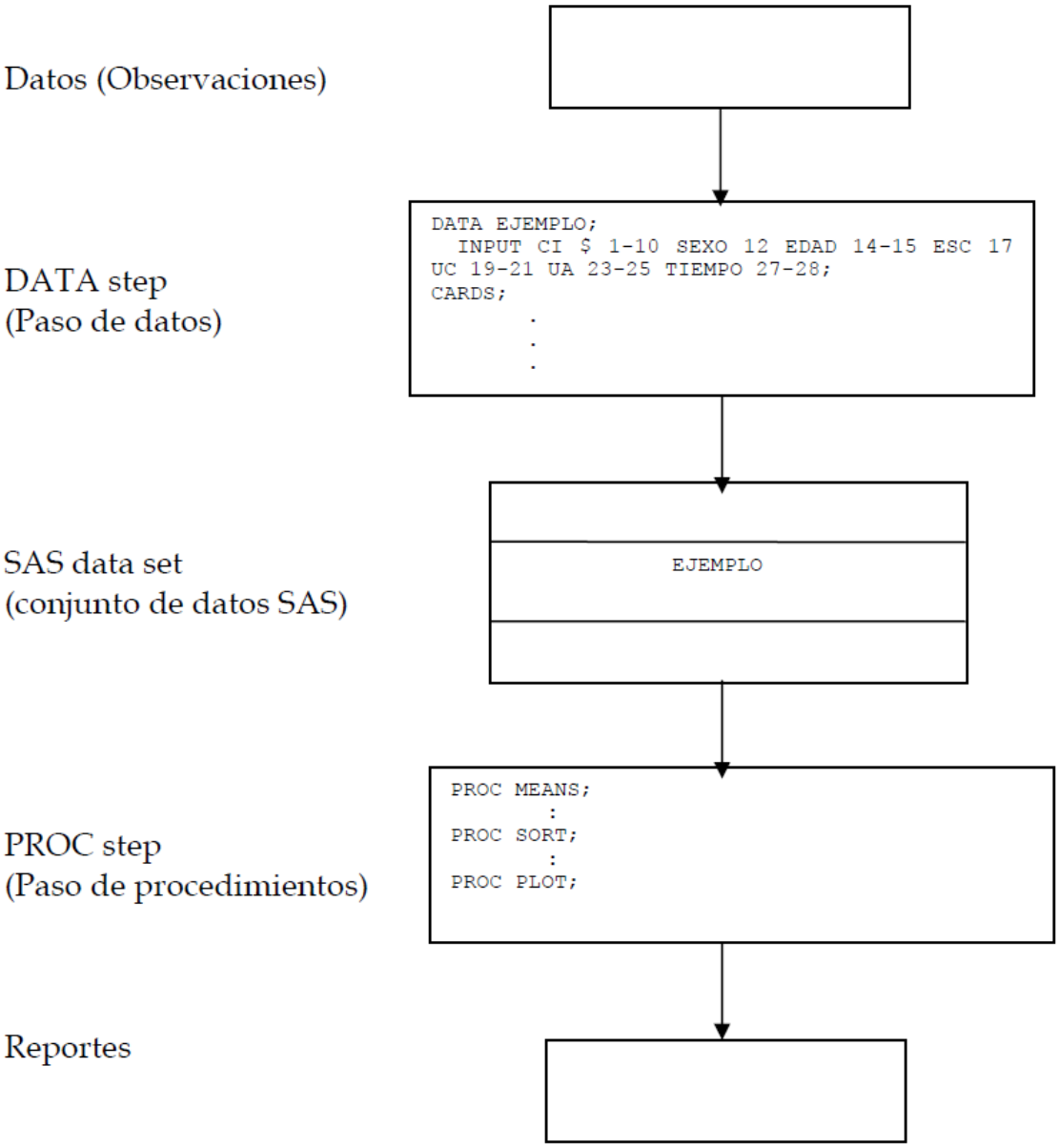


Figura 9: Estructura general de un programa SAS.

El fin de un "step" es marcado por la instrucción RUN, para algunos procedimientos con la instrucción QUIT o por la palabra PROC o DATA que marque el comienzo del siguiente step.

Data sets o conjuntos de datos SAS.

Un “data set” es un conjunto o colección de valores que toman las variables o campos de una entidad (observación) junto con la estructura descriptiva de los mismos, organizados en forma de una matriz rectangular (filas y columnas) y que puede ser reconocida y procesada por el sistema SAS. Una variable es un dato que representa una característica de un objeto de interés para el estudio. Las variables se representan por medio de nombres SAS y pueden ser de dos tipos:

- Variables carácter o alfanuméricas, consisten de cadenas de caracteres, los cuales pueden ser letras, dígitos y caracteres especiales. La longitud máxima de una cadena de caracteres es de 200.
- Variables numéricas, almacenan números que pueden estar precedido por un signo mas (+) o menos (-), representan cantidades numéricas. Están almacenados en formato de punto flotante en 8 bytes, lo que proporcionan espacio para 16 ó 17 dígitos.

La estructura rectangular de un conjunto de datos SAS implica que todas las variables y todas las observaciones tiene que tener un valor. Sin embargo, muchas veces existen valores ausentes. Por defecto, un valor ausente numérico se visualiza como un punto y un alfanumérico se visualiza como un espacio en blanco. La obtención de estos conjuntos de datos implican un procesado de los datos de origen, su esquema se puede ver en la figura 10.

Una observación es un grupo de valores que representan los diferentes atributos o características de un objeto o entidad en estudio, una persona, una región, un país, un almacén, etc. Un "data set", cuya estructura se representa en la figura 11, está constituido por dos áreas:

- Área de descriptores, donde se almacena la documentación del conjunto de datos, es decir, información descriptiva del conjunto de datos y sus variables. Los atributos del conjunto de datos incluyen etiqueta del conjunto de datos, número de observaciones, longitud del registro, fecha de la última modificación. La descripción de las variables incluyen atributos tales como, nombre, tipo, longitud, formato, etiqueta e índice.
- Área de datos, en esta parte se almacenan los datos en forma rectangular (filas y columnas).

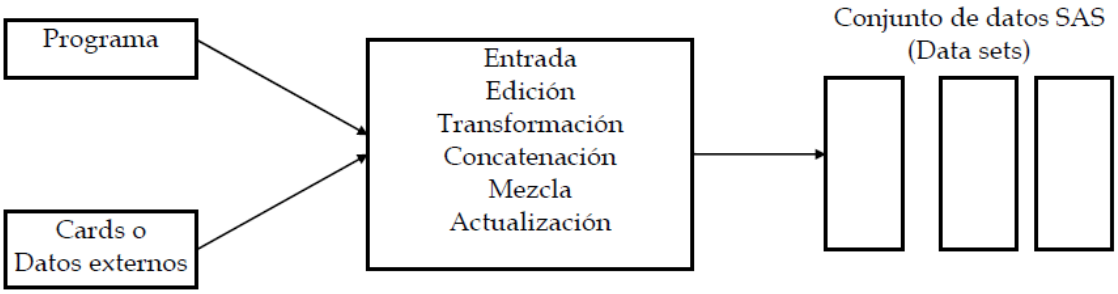


Figura 10: Funcionamiento del paso de datos.

Las columnas en la tabla corresponden a las variables o campos de los datos y las filas las observaciones o registros. El número máximo de variables que se puede crear en un data set es 1024 mientras que en el número de observaciones no hay límite.
Los data set pueden ser almacenados en forma temporal, los cuales se borran o se pierden en el momento de cerrar una sesión de trabajo con el SAS, por lo general se

almacenan en medios permanentes. De esta forma los datos pueden ser utilizados en futuras aplicaciones sin necesidad de volverlos a crear. Cuando se inicia el software SAS se observa una ventana similar a la figura 12.

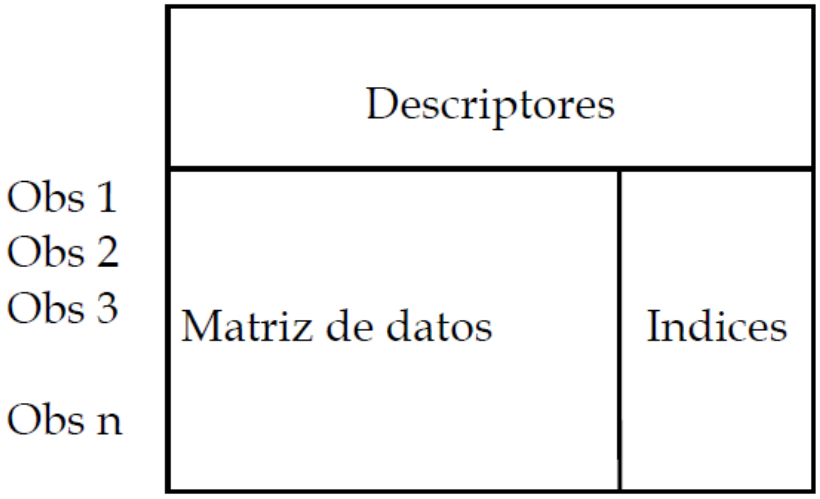


Figura 11: Estructura de un data set.

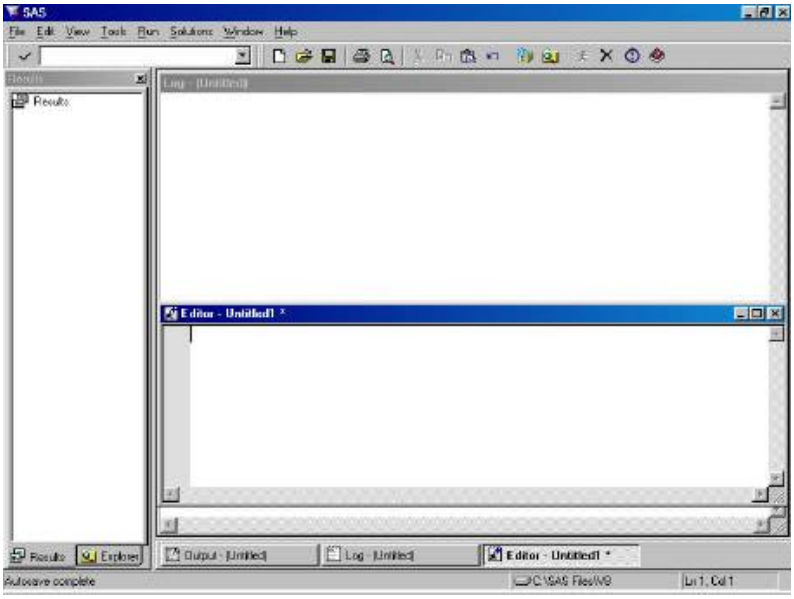


Figura 12: Ventana inicial SAS.

Funciones principales en SAS.

Cuando se trabaja con las variables de la base de datos, es usual generar nuevas variables a partir de aquellas que ya existían utilizando funciones y operadores. Igualmente se emplean funciones y operadores para escribir condiciones utilizando,

además, símbolos para comparar expresiones. Aunque hay una gran cantidad de funciones sólo se describen aquellas que son más usuales:

- FUNCIONES NUMÉRICAS:

ABS(EXPRESIÓN) Valor Absoluto
SQRT(EXPRESIÓN) Raíz Cuadrada
ROUND(EXPRESIÓN) Redondear
ROUND(EXPRESIÓN, PRECISIÓN) Redondear con cierta precisión
EXP(EXPRESIÓN) Exponencial
LOG(EXPRESIÓN) Logaritmo
LOG2(EXPRESIÓN) Logaritmo con base 2
LOG10(EXPRESIÓN) Logaritmo con base 10
COS(EXPRESIÓN) Coseno
SIN(EXPRESIÓN) Seno
TAN(EXPRESIÓN) Tangente

- FUNCIONES ALEATORIAS:

RANBIN(SEMILLA, n, p) Binomial de parámetros "n" y "p" generada a partir de cierta semilla.
RANNOR(SEMILLA) Normal de media "0" y desviación estándar "1" generada a partir de cierta semilla.
RANPOI(SEMILLA, a) Poisson de parámetro "a" generada a partir de cierta semilla.
RANUNI(SEMILLA) Uniforme con parámetros "0" y "1" generada a partir de cierta semilla.

- FUNCIONES ALFANUMÉRICAS:

INDEX Busca una expresión de caracteres dentro de una cadena
COMPRESS Elimina caracteres específicos de una cadena.
LOWCASE Convierte todas las letras del argumento a minúsculas
UPCASE Convierte todas las letras del argumento a mayúsculas
LENGTH Retorna la longitud del argumento
LEFT Alinea a la izquierda una expresión de caracteres
REVERSE Da la vuelta a una cadena
SCAN Selecciona una palabra en particular de una expresión de caracteres
SOUNDEX Codifica una cadena a sonidos para facilitar comparaciones
SPEDIS Determina la similitud entre dos palabras expresada cómo una distancia
SUBSTR Extraes una subcadena de un argumento
TRANSLATE Reemplaza caracteres específicos de un argumento

Exportar/Importar datos en SAS.

```
DATA PACTIVO2;
INFILE 'A:\DATOS.DAT' LRECL=9;
INPUT NUM_PAC 1-2 INIC $ 4-6 SEXO 7 EDAD 8-9;
RUN;

PROC PRINT DATA=PACTIVO2;
RUN;

DATA PACTIVO3;
INFILE 'A:\DATOS2.DAT' DLM='09'x;
INPUT NUM_PAC TRATAM;
RUN;

PROC PRINT DATA=PACTIVO3;
RUN;
```

Figura 13: Importación de un archivo en SAS.

Como se aprecia en la Figura 13, para la lectura de datos externos se utiliza la instrucción INFILE. En ella se menciona la ruta dónde se encuentra el fichero que contiene los datos. La opción LRECL de la instrucción INFILE indica la longitud máxima de cada línea (es indispensable si cada registro tiene más de 256 caracteres).

En la instrucción INPUT se declara las variables que se van a leer. Se escriben las columnas dónde se encuentran las variables si el fichero de datos externo es de formato fijo. En el caso en qué el fichero de datos está delimitado, no tiene sentido especificar las columnas. Por defecto, el separador que lee SAS es el espacio, pero con la opción DLM se define el delimitador deseado, por ejemplo, DLM='09'x si el fichero es encuentra delimitado por tabuladores o DLM=';' si el fichero es encuentra delimitado por el símbolo “; ".

Cualquier procedimiento trabaja con el dataset deseado utilizando la opción DATA=nombre_dataset. Por defecto, SAS utiliza el dataset creado en el paso DATA más reciente.

Es útil observar el listado producido por PROC PRINT para comprobar que efectivamente los datos se han leído perfectamente (en ocasiones no se cometen errores en la sintaxis aunque los datos no se leen adecuadamente).

```
DATA AUXILIAR;
SET FIEBRE.PACTIVO1;
FILE 'A:\DATOS1.DAT' ;
PUT NUM_PAC 1-2 TRATAM 3 INIC $ 4-6 SEXO 7 EDAD 8-9;
RUN;
```

Figura 14: Exportación de un archivo en SAS.

Como se observa en el ejemplo de la Figura 14, la instrucción FILE es necesaria para exportar los datos de un dataset a la ruta y formato deseados, pero sólo sirve para ficheros de tipo texto.

En la instrucción PUT se mencionan las posiciones de cada variable (en el caso que el fichero de datos sea de formato fijo).

Procedimientos SAS.

La sintaxis de los diferentes procedimientos suele ser muy similar. La mayoría de opciones sirven para casi todos los procedimientos, pero con prudencia, ya que cada procedimiento tiene sus particularidades y no siempre estas opciones tienen sentido en cualquier PROC. En líneas generales, la estructura de un procedimiento puede ser similar a la de la figura 15:

```
PROC NOM_PROC DATA=nombre_dataset OPCIONES ESPECÍFICAS;

* INSTRUCCIONES ESPECÍFICAS DEL PROC;
* INSTRUCCIONES OPTATIVAS DEL PROC;

WHERE CONDICIÓN;
BY VARIABLES;
WEIGHT VRAIABLE PESO;
OUTPUT OUT=dataset_salida;
RUN;
```

Figura 15: Estructura de un procedimiento SAS.

A continuación se enuncian los principales procedimientos del componente BASE de SAS:

APPEND Procedure	Añadir datos a un dataset
CALENDAR Procedure	Calendario con fechas y citas
CATALOG Procedure	Manipula los catálogos de SAS
CHART Procedure	Gráficos de barras sencillos
CIMPORT Procedure	Importación de datos de otras versiones de SAS
COMPARE Procedure	Comparación de Bases de datos
CONTENTS Procedure	Contenidos de un dataset
COPY Procedure	Realiza copias de un dataset
CORR Procedure	Correlación entre variables
CPORT Procedure	Exportación de datos de otras versiones de SAS
DATASETS Procedure	Manipulación de datasets (eliminar).
DBCSTAB Procedure	Produce tablas de conversión a caracteres de doble-byte.
DISPLAY Procedure	Visualizar títulos
EXPLODE Procedure	Títulos grandes, vía explotar caracteres.

EXPORT Procedure	Exportar datasets a texto (p. ej.)
FORMAT Procedure	Dar etiquetas a los valores de las variables
FORMS Procedure	Para crear etiquetas adhesivas
FREQ Procedure	Tablas de frecuencias
FSLIST Procedure	Examinar ficheros externos al SAS
IMPORT Procedure	Importar datos (p. ej. en formato texto)
MEANS Procedure	Resumir los datos
OPTIONS Procedure	Opciones de la ventana OUTPUT
PLOT Procedure	Diagramas de dispersión sencillos
PMENU Procedure	Prepara Menús para ser utilizados por otros módulos de SAS
PRINT Procedure	Listar datasets
PRINTTO Procedure	Definir las rutas dónde almacenar las ventanas LOG y OUTPUT
RANK Procedure	Crea Rangos a partir de variables.
REGISTRY Procedure	Mantiene el registro de SAS.
REPORT Procedure	Para realizar informes
SORT Procedure	Ordenar un dataset
SQL Procedure	Consultas a datasets mediante instrucciones SQL
STANDARD Procedure	Produce variables estandarizadas
SUMMARY Procedure	Estadísticos de Resumen
TABULATE Procedure	Tablas
TIMEPLOT Procedure	Diagramas de dispersión respecto el tiempo
TRANSPOSE Procedure	Transpone datasets

2.3. SPSS

El programa SPSS (Statistical Product and Service Solutions), cuyo logotipo se muestra en la figura 16, es un conjunto de potentes herramientas de tratamiento de datos y análisis estadístico. Al igual que el resto de aplicaciones que utilizan como soporte el sistema operativo Windows, SPSS funciona mediante menús desplegables y cuadros de diálogo que permiten hacer la mayor parte del trabajo simplemente utilizando el puntero del ratón.

Figura 16: Logo del software SPSS

SPSS es muy usado en las ciencias sociales y las empresas de investigación de mercado. Originalmente SPSS fue creado como el acrónimo de Statistical Package for the Social Sciences aunque también se ha referido como "Statistical Product and Service Solutions". Sin embargo, en la actualidad la parte SPSS del nombre completo del software (IBM SPSS) no es acrónimo de nada [Ref4, WWW].

Es uno de los programas estadísticos y de análisis de datos más conocidos teniendo en cuenta su capacidad para trabajar con grandes bases de datos y un sencillo interface para la mayoría de los análisis. La versión 12 de SPSS permitía realizar análisis con 2 millones de registros y 250.000 variables. El programa consiste en un módulo base y módulos anexos que se han ido actualizando constantemente con nuevos procedimientos estadísticos. Cada uno de estos módulos se compra por separado como en el caso de la herramienta anteriormente descrita SAS.

SPSS fue creado en 1968 por Norman H. Nie, C. Hadlai Hull y Dale H. Bent. Entre 1969 y 1975 la Universidad de Chicago por medio de su National Opinion Research Center estuvo a cargo del desarrollo, distribución y venta del programa. A partir de 1975 corresponde a SPSS Inc. Originalmente el programa fue creado para grandes computadores. En 1984 sale la primera versión para computadores personales. Desde la versión 14, pero más específicamente desde la versión 15 se ha implantado la posibilidad de hacer uso de las librerías de objetos SPSS desde diversos lenguajes de programación. Aunque principalmente se ha implementado para Python, también existe la posibilidad de trabajar desde Visual Basic, C++ y otros lenguajes. El 28 de junio de 2009 se anuncia que IBM, meses después de ver frustrado su intento de compra de Sun Microsystems, adquiere SPSS, por 1.200 millones de dólares.

Estructura de SPSS.

Al iniciar una sesión con SPSS emerge una ventana de aspecto similar al de una hoja de cálculo: el Editor de datos (ver Figura 17). El Editor de datos es la ventana principal de SPSS, pero no la única. Existen ocho tipos de ventanas SPSS, aunque no todas ellas poseen la misma importancia desde el punto de vista de su utilidad para el usuario. Las dos ventanas principales (imprescindibles para trabajar con SPSS) son:

- El Editor de datos. Contiene el archivo de datos sobre el que se basa la mayor parte de las acciones que es posible llevar a cabo con el SPSS. El Editor de datos se abre automáticamente (vacío, sin datos: Figura 17) cuando se entra en SPSS. La ventana del Editor de datos puede mostrar dos contenidos diferentes: los datos propiamente dichos y las variables del archivo acompañadas del conjunto de características que las definen. Al igual que el resto de ventanas SPSS, el Editor de datos contiene una barra de menús (un conjunto de menús desplegables), una barra de herramientas (una serie de botones-iconos que facilitan el acceso rápido a muchas de las funciones SPSS) y una barra de estado (con información puntual sobre diferentes aspectos relacionados con el estado del programa). Es posible abrir más de un Editor de datos y, por tanto, trabajar

con varios archivos de datos simultáneamente. No obstante, los datos que interese analizar juntos deberán estar en el mismo archivo. En este editor se representa la estructura de datos fundamental de SPSS. Las filas representan los casos y las columnas representan las variables. Cada casilla tiene un valor que corresponde a un determinado caso en una determinada variable.

- El Visor de resultados. Recoge toda la información (estadísticos, tablas, gráficos, etc.) que SPSS genera como consecuencia de las acciones que lleva a cabo. El Visor permite editar los resultados y guardarlos para su uso posterior. Es posible tener abiertas varias ventanas del Visor asociadas a cada Editor de datos.

Los resultados del Visor adoptan tres formatos distintos: tablas, gráficos y texto. SPSS dispone de un editor (y, por tanto, una ventana distinta) para cada uno de estos tres formatos básicos. El resto de ventanas son el borrador del Visor de resultados, el editor de sintaxis y el editor de procesos.

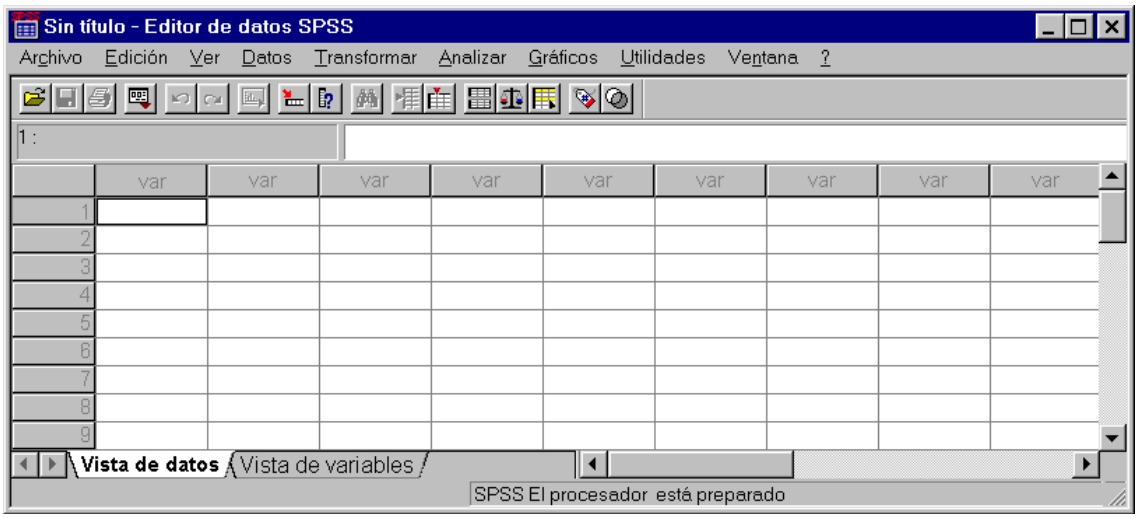


Figura 17: Ventana del editor de datos de SPSS.

Tareas usuales con SPSS.

En una sesión estándar para analizar ficheros, se suelen realizar cuatro tareas básicas con el software SPSS:

- Abrir un archivo de datos.
- Ejecutar un procedimiento estadístico.
- Examinar los resultados.
- Exportar los resultados.

Abrir un archivo de datos.

Una vez abierto el Editor de datos (ver Figura 17), la primera acción suele consistir en

introducir datos desde el teclado o en abrir un archivo de datos existente. Para ello se selecciona la opción Abrir del menú Archivo para acceder al cuadro de diálogo Abrir archivo (ver Figura 18). Este cuadro de diálogo muestra, por defecto, un listado de los archivos con extensión .sav (archivos de datos en formato SPSS).

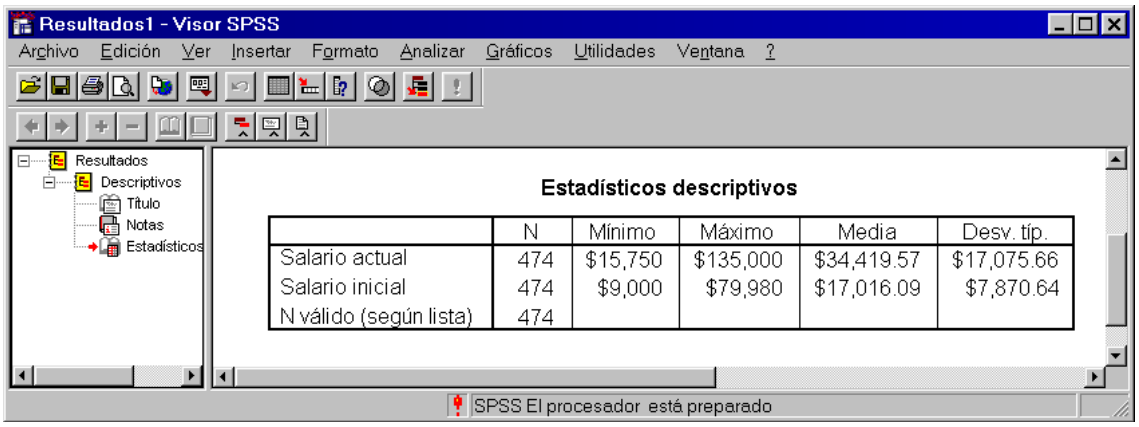


Figura 18: Menú Archivo SPSS.

Al abrir un archivo de datos, el Editor de datos, hasta ahora vacío, toma el aspecto que muestra la Figura 19.

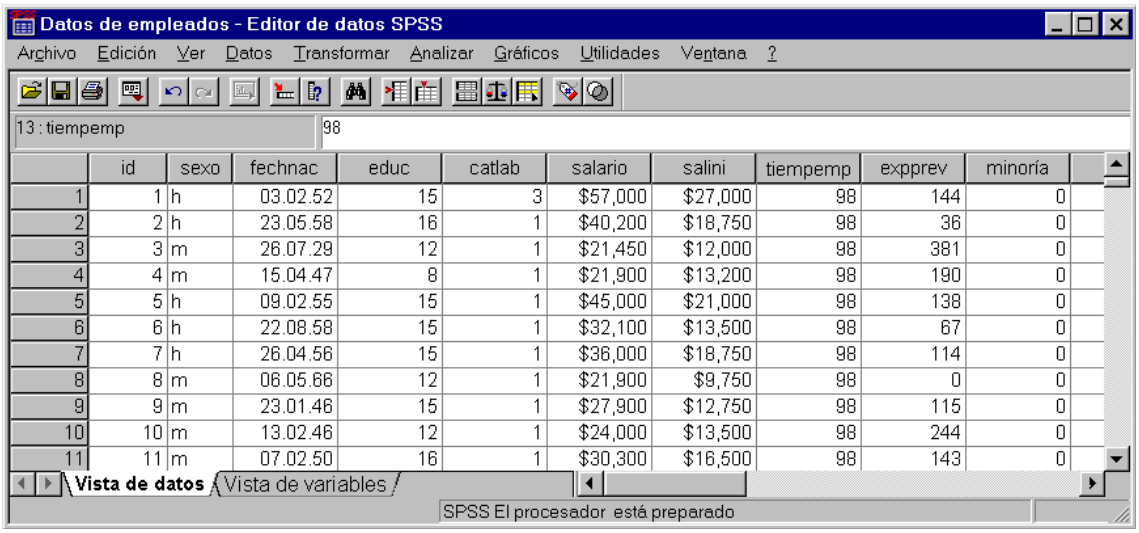


Figura 19: Editor de datos.

Ejecutar un procedimiento estadístico.

Una vez abierto un archivo de datos, la segunda acción que suele llevarse a cabo en una sesión con SPSS consiste en seleccionar algún procedimiento estadístico.

Para ello se selecciona el menú Analizar. Dentro de este menú, como se muestra en la

figura 20, se encuentran multitud de procedimientos estadísticos. Por ejemplo, si se selecciona Estadísticos descriptivos> Frecuencias se podrá generar una tabla de frecuencias (ver Figura 21).

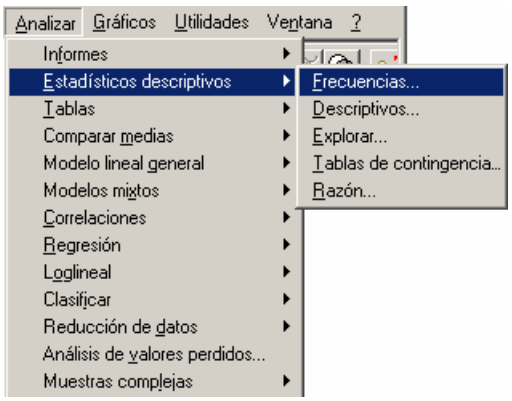


Figura 20: Generación de tabla de frecuencias con SPSS.

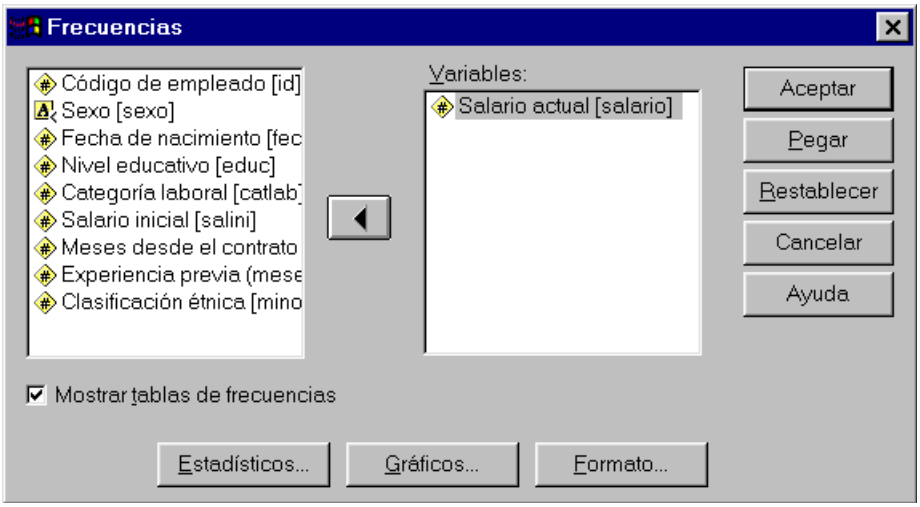


Figura 21: Cuadro diálogo de Frecuencias.

Examinar los resultados.

El Visor de resultados es el encargado de recoger la información que SPSS genera como consecuencia de los procedimientos que ejecuta. Puesto que únicamente se ha solicitado un histograma de la variable salario, el Visor de resultados contiene únicamente el histograma que muestra la figura 22.

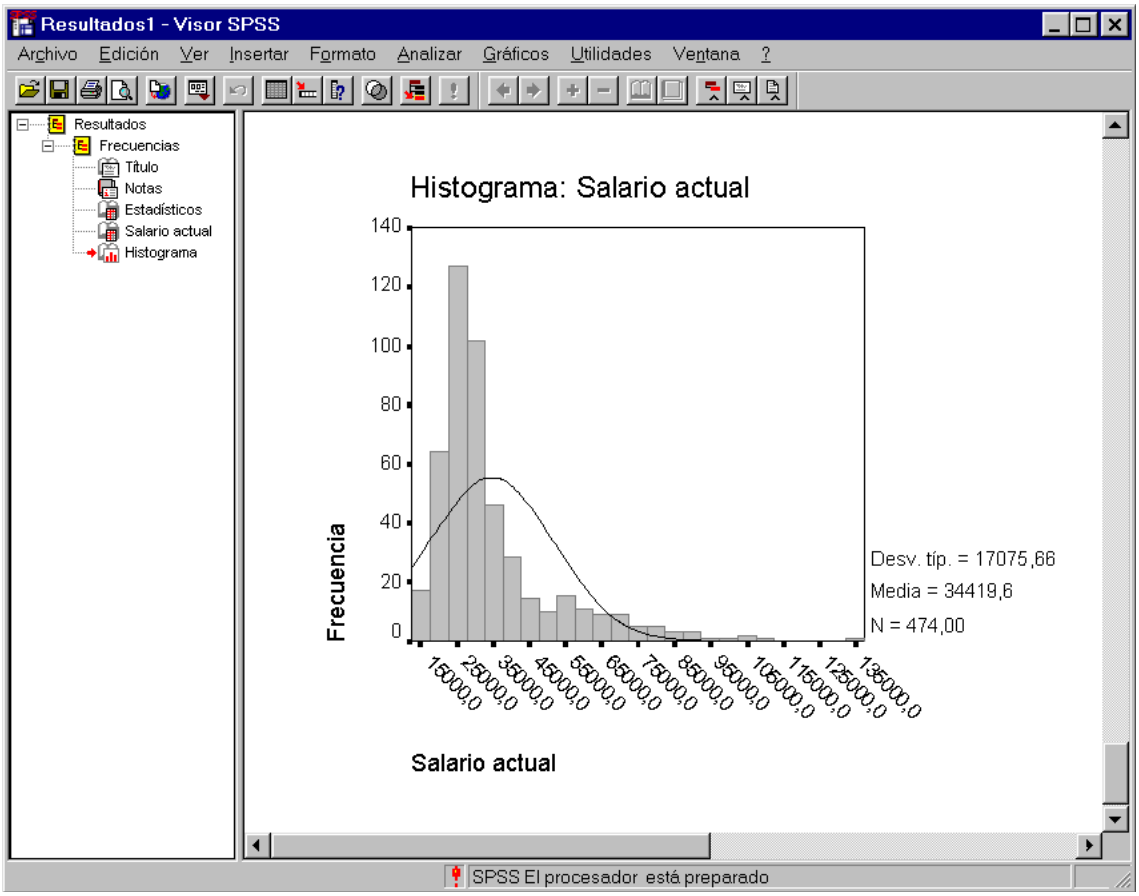


Figura 22: Visor de resultados con tabla de frecuencias de variable Salario.

Exportar los resultados.

La opción Exportar permite guardar los objetos de texto y las tablas pivotantes en formato HTML y en formato de texto ASCII. Además permite guardar los gráficos en una amplia variedad de formatos. Para exportar total o parcialmente el archivo de resultados se selecciona la opción Exportar del menú Archivo. El aspecto visual de menú se muestra en la figura 23.

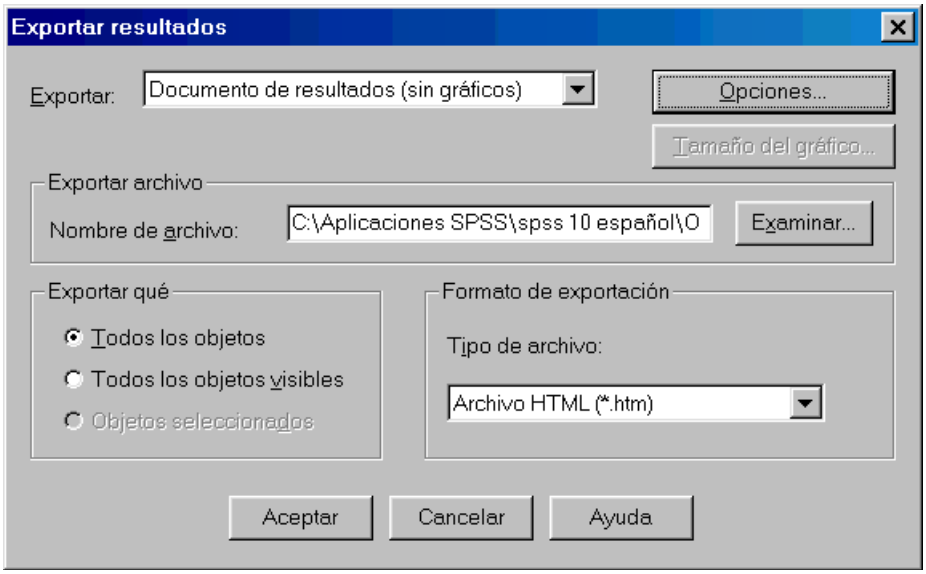


Figura 23: Cuadro dialogo Exportar resultados.

Este menú contiene tres opciones (pulsando el botón de menú desplegable) para decidir qué parte del archivo de resultados se desea exportar: documento completo, documento sin gráficos y sólo los gráficos.

2.4. MongoDB

MongoDB, cuyo logotipo se muestra en la figura 24, es un sistema de base de datos distribuido NoSQL y orientado a documentos. Esto quiere decir que en lugar de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Figura 24: Logo MongoDB.

El desarrollo de MongoDB empezó en octubre de 2007 por la compañía de software 10gen. Ahora MongoDB es una base de datos lista para ser utilizada y con muchas características. Las propiedades más resaltables de MongoDB son su velocidad y su rico pero sencillo sistema de consulta de los contenidos de la base de datos. Se podría decir que alcanza un balance perfecto entre rendimiento y funcionalidad, incorporando muchos de los tipos de consulta que utilizaríamos en nuestro sistema relacional, pero sin sacrificar en rendimiento. Además de estas dos características, esta tecnología presenta una alta escalabilidad. MongoDB es una base de datos distribuida y fue diseñada para que ser altamente escalable. Su modelo de datos orientada a documentos permite que sea más fácil distribuir datos a través de múltiples servidores o computadoras, como se esquematiza en la figura 25. MongoDB se encarga automáticamente de mantener el equilibrio de datos y carga de computación a lo largo del clúster, de la redistribución automática de los documentos y de enrutar las peticiones de usuario a las maquinas correctas, lo que permite a los desarrolladores centrarse únicamente en el desarrollo de las aplicaciones. En el momento que un clúster necesita mayor capacidad, se puede añadir nuevas maquinas y MongoDB se percatará automáticamente de como los datos deben ser distribuidos entre las nuevas computadoras incorporadas. Esta tecnología es de código abierto y se puede descargar desde su web oficial [Ref5, WWW].

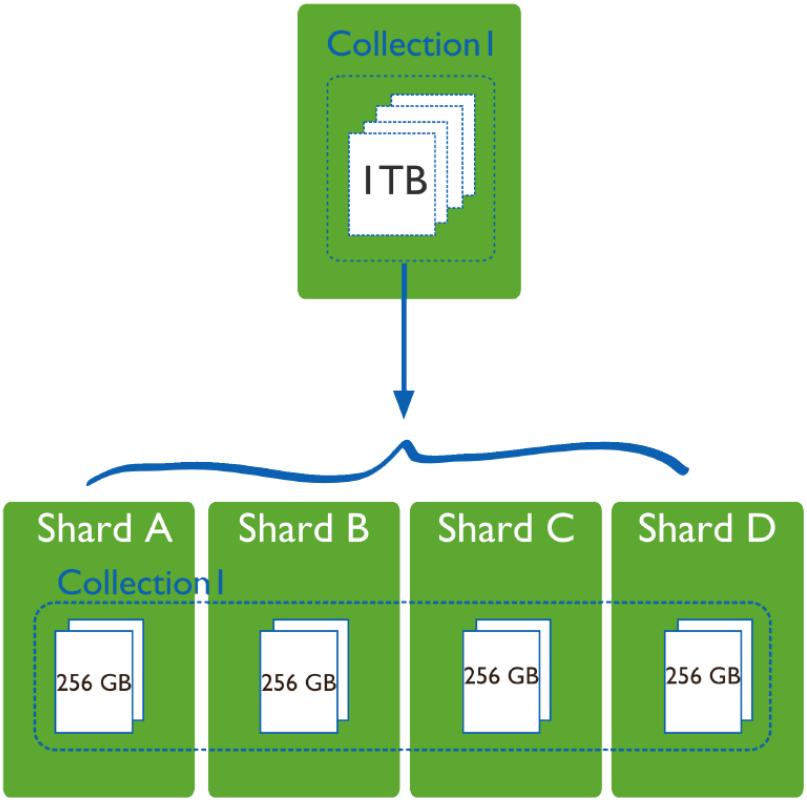


Figura 25: Distribución de datos a través de un clúster MongoDB.

Esta base de datos es altamente utilizada en las industrias y MTV Network, Craigslist y Foursquare son algunas de las empresas que utilizan esta base de datos.

Formato de los documentos.

En MongoDB, cada registro o conjunto de datos se denomina documento. Los documentos se pueden agrupar en colecciones, las cuales se podría decir que son el equivalente a las tablas en una base de datos relacional (sólo que las colecciones pueden almacenar documentos con formatos muy distintos, en lugar de estar sometidos a un esquema fijo). Se pueden crear índices para algunos atributos de los documentos, de modo que MongoDB mantendrá una estructura interna eficiente para el acceso a la información por los contenidos de estos atributos.

Los distintos documentos se almacenan en formato BSON, o Binary JSON, que es una versión modificada de JSON que permite búsquedas rápidas de datos. BSON guarda de forma explícita las longitudes de los campos, los índices de los arrays y demás información útil para el escaneo de datos. Esto es debido a que, en algunos casos, el mismo documento en BSON ocupa un poco más de espacio de lo que ocuparía de estar almacenado directamente en formato JSON. Pero una de las características claves en los sistemas NoSQL es que el almacenamiento es económico (clúster de computadoras convencionales) y es mejor aprovecharlo si de este modo se introduce un considerable incremento en la velocidad de localización de información dentro de un documento.

No obstante, en la práctica, no se ve el formato en que verdaderamente se almacenan los datos, y se trabaja siempre sobre un documento en JSON tanto al almacenar como al consultar información. Un ejemplo de un documento en MongoDB podría ser el siguiente, que representa el modo en el que podrían almacenarse los posts de un blog:

```
{
    "_id"          : "4da2c0e2e999fb56bf000002"
    "title"        : "Una introducción a MongoDB",
    "body"         : "Lorem ipsum dolor sit amet…",
    "published_at" : "2011-05-09T18:17:07-07:00",
    "author_info"  : {
                       "_id"  : "4dc8919331c0c00001000002"
                       "name" : "Carlos Paramio"
                     },
    "tags"         : ["MongoDB", "NoSQL", "Bases de datos"]
    "comments"     : [
                       {
                         "author_info" : { "name" : "Jorge Rubira",
"email" : "email1@example.com" },
                         "body"        : "Test",
                         "created_at"  : "2014-05-10T10:14:01-07:00"
                       },
                       {
                         "author_info" : { "name" : "Francisco
Rodríguez", "email" : "email2@example.com" },
                         "body"        : "Otro test",
                         "created_at"  : "2014-05-10T10:14:09-07:00"
                       }
                     ]
    "liked_by"     : ["4d7cf768e999fb67c0000001",
"4da34c62ba875a19d4000001"]
```

```
}
```

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes. Es decir, se podría formar una colección con el documento anterior y otro documento que no contiene los mismos atributos.

Como se puede observar en el primer documento, se pretende representar la manera en que podrían almacenarse los datos correspondientes a un post de un blog. Los atributos "_id" (o clave principal) pueden tener el formato que se desee, aunque MongoDB utiliza un valor parecido a un UUID en hexadecimal por defecto si no se ha especificado ninguno. A pesar de parecer un valor completamente aleatorio (aunque es sabido que la aleatoriedad real no existe en informática), utilizan como base una semilla basada en la MAC de la interfaz de red de la máquina (y otros detalles de la misma) para evitar que dos máquinas diferentes puedan generar el mismo valor para la clave de un documento. Y los primeros bytes corresponden a una marca de tiempo, de modo que las claves se ordenan de forma natural por orden de creación (o casi, pues está claro que las distintas máquinas MongoDB deben tener la fecha y hora sincronizadas) sin tener que mirar cuál fue el último valor usado.

Las etiquetas y los comentarios están en el propio documento, en lugar de guardarlos en colecciones separadas y utilizar claves foráneas(o clave ajena) para referenciar a los mismos. Sin embargo, en el atributo "liked_by" sí que guardamos una relación de claves, que corresponden a los usuarios que han marcado el post como que les ha gustado. Utilizar una forma u otra dependerá de las necesidades de acceso a estos datos. En este caso, por ejemplo, no se va a mostrar información sobre los usuarios que han marcado un post con un "me gusta", pero sí se va a ver cuántos lo han marcado así o si el usuario actual ya lo ha marcado o no, con lo que almacenar únicamente las claves de esos usuarios y guardar su información personal detallada en otra colección es lo más conveniente.

Por supuesto, no es necesario pedir a MongoDB que devuelva todo el documento cada vez que lo consultamos. Si se va a mostrar únicamente un listado de posts recientes, seguramente sea suficiente obtener el atributo "title", con los documentos ordenados por "published_at". De este modo se ahorra ancho de banda entre el motor de base de datos y la aplicación al mismo tiempo que memoria, dado que no hay que instanciar todo el documento. Además, si se tienen muchos miles de visitantes, el atributo "liked_by" podría llegar a crecer bastante. Aunque el tamaño de un documento de MongoDB puede llegar hasta los 16 Megabytes, con lo que se puede almacenar bastante información dentro de un único documento sin necesidad de utilizar referencias. En caso de que se tuviera que almacenar mucho más, habría que optar por utilizar otro esquema.

En ocasiones es necesario desnormalizar para poder tener a mano la información necesaria a la hora de mostrar un post. Es por eso que en el atributo "author_info" se ha utilizado una versión intermedia: Si bien se tiene la clave principal del usuario que

ha escrito este post, como es habitual que se muestre el nombre del autor, se ha almacenado también dicho nombre en el documento que representa al post, para que no sea necesario realizar una segunda consulta a la colección "usuarios". Estas desnormalizaciones dependen nuevamente del uso que se den a los datos. En este caso, es evidente que el nombre de un usuario no va a variar demasiado, así que recorrer todos los posts para cambiar este valor en caso de que el usuario realice esta operación, si bien es una modificación que podría llevar un tiempo considerable para ejecutarse, no es una operación habitual frente a la consulta del nombre del autor, y por tanto, compensa. Incluso se podría llegar a tratar el post como algo más permanente, de modo que aunque un usuario cambiara su nombre a posteriori, el nombre utilizado para firmar los posts anteriores no varíe o sencillamente los posts puedan firmarse con diferentes pseudónimos.

Como se puede intuir, el modelado del esquema de datos con MongoDB depende más de la forma en que se consultarán o actualizarán los datos que de las limitaciones del propio sistema. Un esquema para un ejemplo similar al explicado se muestra en la figura 26.

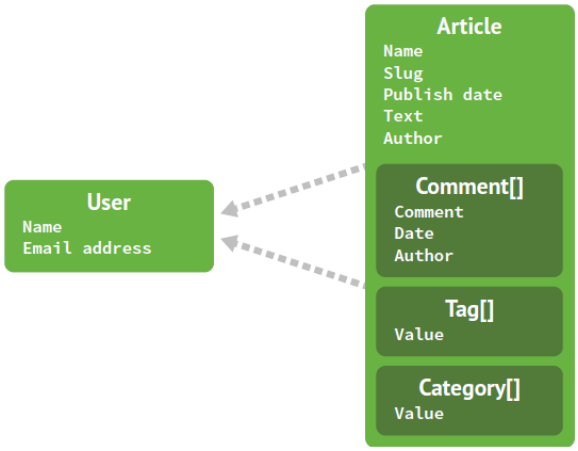


Figura 26: Ejemplo de modelo de documento para un blog

Como consultar los datos.

Sin entrar demasiado en detalles acerca de todas las posibilidades que MongoDB ofrece para consultar los datos almacenados, sí se van a citar algunas de ellas, para señalar que no se está frente a un sistema simple de almacenamiento de pares clave-valor.

En primer lugar, MongoDB permite utilizar funciones Map y Reduce escritas en Javascript para seleccionar los atributos que interesan de los datos, y agregarlos (unificarlos, simplificarlos) del modo deseado, respectivamente. Esto es algo habitual en muchos sistemas NoSQL, y en algunos casos es incluso la única forma posible de consultar datos. Aunque muchas veces necesitamos algo bastante más sencillo que esto.

En MongoDB se pueden utilizar consultas al valor de un atributo específico. Por ejemplo, se puede capturar el post que tiene un determinado título:

```
db.posts.find({'title' : 'Explicación de MongoDB'})
```

El valor a consultar puede estar anidado en un tipo de datos más completo en el atributo del documento (por ejemplo, como valor de una tabla hash asociada al atributo, o como el valor de uno de los ítems de un array). Se utiliza un punto como separador de los nombres de las claves de los diferentes hashes que hay que recorrer hasta llegar al valor deseado. Por ejemplo, la siguiente consulta devolvería todos los posts escritos por un determinado autor:

```
db.posts.find({'author_info._id' : '4da2c0e2e999fb56bf000002'})
```

Y esta otra los posts etiquetados con MongoDB:

```
db.posts.find({'tags' : 'MongoDB'})
```

El hash utilizado como conjunto de condiciones que deben cumplir los documentos a devolver puede incluir operadores de muy diversos tipos, no sólo comparadores del valor absoluto buscado. Algunos de ellos son:

- $all : Para indicar que el array almacenado como valor del atributo debe tener los mismos elementos que el proporcionado en la condición.
- $exists : Para comprobar que el atributo existe en el documento.
- $mod : Para comprobar el resto de una división del valor del atributo por un número.
- $ne : Para indicar que el valor no puede ser el proporcionado.
- $in : Para indicar que el valor debe estar entre alguno de los proporcionados.
- $nin : Contrario de $in.
- $or : Para indicar que se debe cumplir al menos una condición de entre un grupo de condiciones.
- $nor : Contrario de $or.
- $size : Para indicar el número de elementos que debe haber en el array almacenado como valor.
- $type : Para comprobar el tipo del valor almacenado (número, cadena…)

Cuando utilizar MongoDB.

Aunque se suele decir que las bases de datos NoSQL tienen un ámbito de aplicación reducido, MongoDB puede ser utilizado en muchos de los proyectos que desarrollamos en la actualidad.

Cualquier aplicación que necesite almacenar datos semi estructurados puede usar MongoDB. Es el caso de las típicas aplicaciones CRUD o de muchos de los

desarrollos web actuales.

No obstante, aunque las colecciones de MongoDB no necesitan definir un esquema, es importante diseñar la aplicación para seguir uno. Se tendrá que pensar si se necesita normalizar los datos, denormalizarlos o utilizar una aproximación mixta. Estas decisiones pueden afectar al rendimiento de la aplicación. El propio esquema lo definen las consultas que vayamos a realizar con más frecuencia.

MongoDB es especialmente útil en entornos que requieran escalabilidad. Con sus opciones de replicación, se puede conseguir un sistema que escale horizontalmente sin problemas. Esto provoca que MongoDB se adapte perfectamente a aplicaciones de Big Data donde la cantidad de datos es continuamente creciente y suelen no estar estructurados.

MongoDB & Hadoop.

Como en el caso de la plataforma estadística R, previamente explicada, MongoDB es combinable con Hadoop ya que pueden utilizarse juntos para abordar complejos análisis y procesamiento de datos para ficheros almacenados en MongoDB.

El conector de MongoDB para Hadoop es un plugin para Hadoop que proporciona la habilidad de usar MongoDB como entrada o salida de datos. Como los dos software son de código libre podemos descargar el plugin desde la página oficial de MongoDB [Ref6, WWW]. El esquema de uso de MongoDB dentro de Hadoop se muestra en la figura 27.

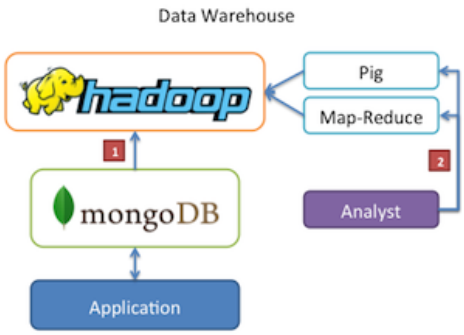


Figura 27: Esquema de uso de MongoDB en un entorno Hadoop.

2.5. Disco project

Disco, cuyo logo se muestra en la figura 28, es una implementación de MapReduce para programación distribuida. Soporta computación paralela sobre grandes conjuntos de datos y almacenamiento seguro en clúster de computadoras. Esto hace que sea una herramienta perfecta para el análisis y el procesamiento de grandes conjuntos de

datos, sin tener que preocuparse por aspectos técnicos relacionadas con la distribución, tales como protocolos de comunicación, equilibrio de carga, el bloqueo, la planificación de tareas, y la tolerancia a fallos, que son manejados por Disco.

Esta plataforma se puede utilizar para una variedad de tareas relativas al análisis de datos: análisis a gran escala, la construcción de modelos probabilísticos, algoritmos de machine learning y muchas otras tareas.

Figura 28: Logo de Disco project.

Este proyecto supone una re-implementación de MapReduce a través de un enfoque distinto al de Hadoop. Disco utiliza una combinación de Python y Erlang lo que trae a un framework MapReduce herramientas basadas en Python.

Disco nació en el Nokia Research Center en 2008 para resolver desafíos reales en el manejo de grandes cantidades de datos. Desde entonces se ha desarrollado activamente por Nokia y muchas otras compañías. Se trata de un proyecto de código libre que puede ser descargado desde su web oficial [Ref7, WWW].

Características de Disco.

El núcleo de Disco está escrito en Erlang, un lenguaje funcional diseñado para construir aplicaciones distribuidas con tolerancia a fallos. Los usuarios de Disco normalmente escriben trabajos en Python, lo que hace posible expresar algoritmos más complejos con muy poco código.

Por ejemplo, el siguiente ejemplo totalmente funcional calcula frecuencias de palabras en un texto de gran tamaño, este ejemplo se realizará en el siguiente capítulo con Hadoop:

```
from disco.core import Job, result_iterator

def map(line, params):
    for word in line.split():
        yield word, 1

def reduce(iter, params):
    from disco.util import kvgroup
    for word, counts in kvgroup(sorted(iter)):
        yield word, sum(counts)
```

```
if __name__ == '__main__':
    job =
Job().run(input=["http://discoproject.org/media/text/chekhov.txt"],
                  map=map,
                  reduce=reduce)
    for word, count in result_iterator(job.wait(show=True)):
        print(word, count)
```

Esta tecnología, está meditada para integrarse fácilmente en aplicaciones grandes por lo que las tareas computacionalmente exigentes se pueden delegar a un grupo de computadoras independientes de la aplicación principal. Disco proporciona una API de Phyton bastante compacta (normalmente sólo se necesitan dos funciones). Al iniciar el software se puede observar una ventana similar a la de la figura 29.

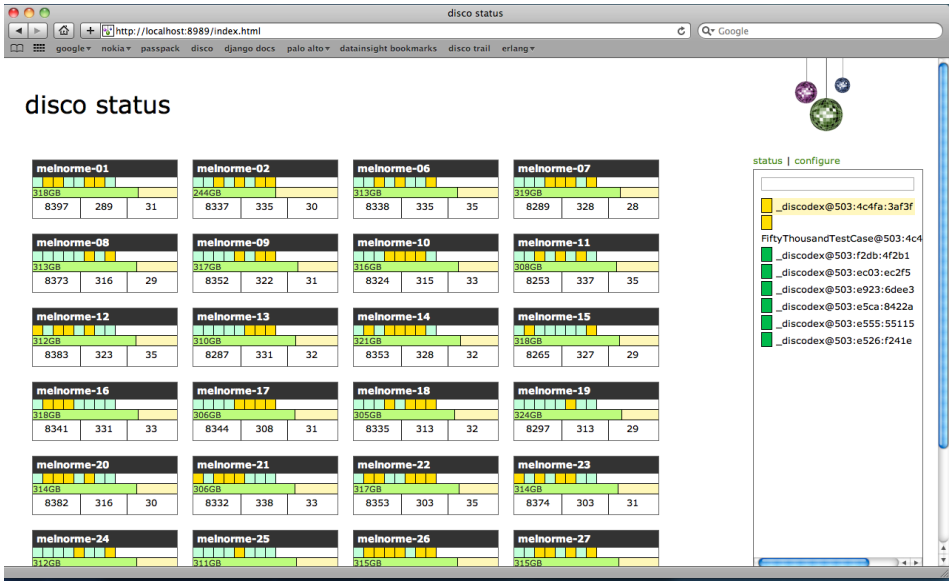


Figura 29: Pantalla principal de Disco.

A continuación se enuncian las características más importantes de Disco:

- Alta escalabilidad: Disco permite su ejecución en un clúster de computadoras y la realización de tareas simultáneamente. Asimismo si necesitamos incrementar la capacidad del sistema, es muy sencillo añadir nuevas computadoras.

- Fácil de usar: Las tareas típicas consiste en dos funciones escritas en Python y dos llamadas a la API de Disco. Al contrario de Disco, la programación MapReduce para Hadoop realizada en JAVA puede ser bastante compleja.

- Las tareas se pueden especificar en otros lenguajes de programación mediante la implementación del protocolo Disco.

- Los datos de entrada pueden estar en cualquier formato. Además los datos

pueden estar situados en cualquier fuente que sea accesible a través de HTTP o pueden estar distribuidos en discos locales.

- Tolerancia a fallos: El bloqueo de un servidor no interrumpe las tareas que estén en ejecución. Se pueden añadir nuevos computadores sobre la marcha.

- Flexible: Además de las funciones map y reduce el usuario puede proporcionar funciones de combinación o de lectura de entradas.

- Disco provee un sistema de almacenamiento distribuido (Disco Distributed FileSystem).

Arquitectura de Disco.

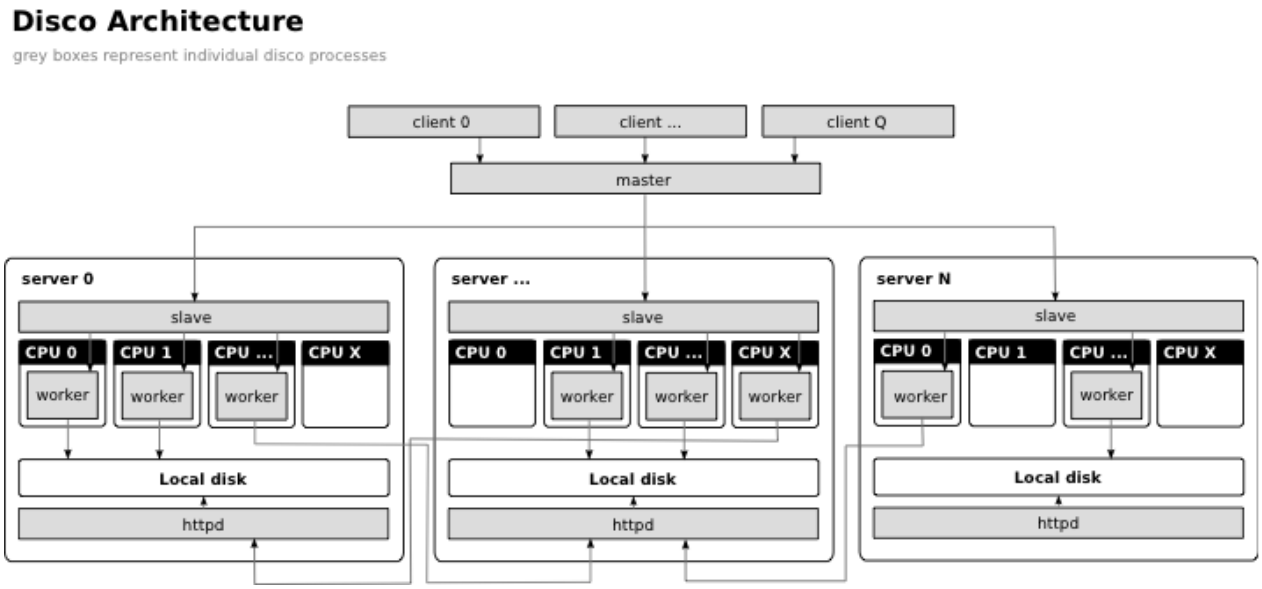


Figura 30: Arquitectura Disco.

Disco, como se puede distinguir en la figura 30, está basado en la arquitectura maestro-esclavo:

- El maestro recibe los trabajos, los agrega a la cola de trabajos y los ejecuta en el clúster cuando los nodos están disponibles.

- Los programas cliente envían trabajos al maestro.

- Los esclavos son iniciados por el maestro en cada nodo del clúster. Se monitorizan todos los procesos que se ejecutan en sus respectivos nodos.

- Los workers realizan la computación y sus resultados se envían al maestro.

- Una vez que los archivos se almacenan en el clúster Disco intenta mantener los datos localizados mediante la programación de tareas que usan esos archivos como entrada en los mismos nodos en los que se almacenan. Disco ejecuta un servidor HTTP en cada nodo para que los datos sean accesibles remotamente cuando un worker no se puede ejecutar en el mismo nodo donde se encuentran los datos.

Los usuarios pueden limitar el número de workers que se ejecutan en paralelo en cada nodo. Por lo tanto, pueden fácilmente ejecutar tantas tareas como CPUs hay disponibles en el clúster.

Disco permite designar más de un máster para conseguir una alta disponibilidad y no permitir que tengamos un único punto de fallos.

Disco Distributed Filesystem (DDFS).

El DDFS proporciona a Disco un almacenamiento distribuido de los datos. Está diseñado específicamente para apoyar los casos de uso que son típicos para Disco y MapReduce en general: almacenamiento y procesamiento de grandes cantidades de datos. Esto hace que sea propicio para almacenar datos de registro, objetos binarios (fotos, vídeos, índices) o colecciones de datos brutos.

En este sentido, DDFS es complementario a las bases de datos relacionales tradicionales o almacenamientos clave-valor distribuidos que a menudo tienen dificultades para ampliar a tera o petabytes de datos. Aunque DDFS no es una base de datos de propósito general. Más bien, se trata de una capa de almacenamiento de propósito específico similar al sistema de archivos de Google o Hadoop.

DDFS es un componente de bajo nivel en la pila de Disco, encargado de la distribución de datos, la replicación, el direccionamiento y el acceso. No proporciona una sofisticada consulta de datos en sí misma, sino que está integrada con los trabajos que se realizan en Disco. Como almacenar los resultados provenientes del análisis, proporcionando persistencia y fácil acceso a los datos tratados.

DDFS es un sistema de archivos basado en etiquetas: En lugar de tener que organizar los datos en directorios jerárquicos, permite etiquetar conjuntos de objetos con nombres arbitrarios y recuperarlos más adelante sobre la base de las etiquetas dadas. Esto proporciona una manera flexible para gestionar grandes cantidades de datos. Además proporciona un mecanismo para almacenar atributos arbitrarios con las etiquetas, por ejemplo, para indicar el tipo de datos.

Este componente, no sigue un esquema, lo que permite almacenar datos arbitrarios, no normalizados. Sin embargo, no es adecuado para el almacenamiento de elementos

de datos muy pequeños (menos de 4K) o que necesitan ser actualizados a menudo, tales como contraseñas de usuario o indicadores de estado. Puede almacenar datos en constante cambio en un almacén de claves-valor o una base de datos relacional. Si fuera necesario analizar estos datos con Disco, se puede volcar una copia de la base de datos completa, por ejemplo, para actualizar sus modelos de usuario cada noche.

Además DDFS es altamente escalable. Los nodos nuevos se pueden añadir a la agrupación de almacenamiento sobre la marcha, usando la interfaz de usuario de web Disco. Únicamente, los metadatos se manejan centralmente, lo que garantiza que se mantienen constantes todo el tiempo.

Este sistema de archivos, al igual que MongoDB, está diseñado para funcionar en hardware convencional. La tolerancia a fallos y alta disponibilidad son asegurados mediante la replicación de los datos y metadatos.

2.6. Storm

Durante la última década se ha producido una revolución en el tratamiento de datos. MapReduce, Hadoop, y las tecnologías relacionadas han hecho posible el almacenamiento de datos y su procesamiento a escalas antes impensables. Por desgracia, estas tecnologías de procesamiento de datos no son sistemas en tiempo real, ni están destinados a serlo. No es posible convertir Hadoop en un sistema en tiempo real ya que el procesamiento de datos en tiempo real tiene diferentes requisitos de los requeridos por el procesamiento por lotes.

Figura 31: Actual logo de Apache Storm

Sin embargo, el procesamiento de datos en tiempo real a escala masiva es cada vez más esencial para las empresas. La falta de un "Hadoop de tiempo real" se ha convertido en el mayor agujero en el ecosistema de procesamiento de datos. Storm se crea con el objetivo de intentar cubrir esa carencia.

Apache Storm, cuyo logotipo se muestra en la figura 31, es un sistema distribuido de computación en tiempo real de código libre y abierto. Similar a Hadoop, proporciona un conjunto de primitivas generales para realizar procesamiento por lotes (batch processing), Storm pauta un conjunto de condiciones globales para poder realizar el

cálculo en tiempo real. Se trata de un framework sencillo, multiplataforma y se puede manejar con distintos lenguajes de programación.

Storm fue creado por Nathan Marz y el equipo de BackType, el proyecto fue abierto cuando Twitter lo adquirió. En 2013 la fundación Apache Software aceptó a Storm en su programa de incubación.

Características de Storm.

Storm impone un conjunto de principios para poder realizar el cálculo en tiempo real. Del mismo modo que MapReduce facilita enormemente la escritura de procesamiento por lotes en paralelo, las primitivas de Storm facilitan enormemente la escritura de la computación en paralelo en tiempo real.

Las propiedades fundamentales de Storm son:

- Amplia utilidad: Storm puede ser utilizado para el procesamiento de mensajes, actualización de bases de datos (procesamiento de flujo), realización de consultas de datos continuas y envío de los resultados al cliente (computación continua), para la paralelización de una consulta intensa como una consulta de búsqueda sobre la marcha (RPC distribuida) y muchas otras aplicaciones.

- Alta escalabilidad: Para ampliar una topología, basta con agregar máquinas y aumentar la configuración de paralelismo de la topología.

- Garantiza que no se pierden datos: Un sistema en tiempo real debe tener amplias garantías de que los datos van a ser procesados exitosamente ya que sistema que permite la perdida de datos, tiene un conjunto de casos de uso muy limitado.

- Muy robusto: A diferencia de sistemas como Hadoop, que son conocidos por ser difíciles de utilizar, las agrupaciones Storm son fácilmente manejables. Es un objetivo explícito del proyecto Storm, facilitar al usuario la gestión de estas agrupaciones.

- Tolerante a fallos: En caso de anomalías durante la ejecución de sus cálculos, Storm volverá a asignar tareas según sea necesario. La plataforma se asegura de que un cálculo puede ejecutarse por siempre (o hasta que el usuario mata el proceso).

Componentes de un clúster Storm.

Un clúster Storm es aparentemente similar a un clúster Hadoop. Considerando que, sobre Hadoop se ejecutan "trabajos MapReduce", y en Storm "topologías". Los "trabajos (jobs en Hadoop)" y "topologías" en sí son muy diferentes. Una diferencia clave es que un Job MapReduce termina mientras que una topología procesa los datos de entrada siempre (o hasta que el usuario mata el proceso).

Hay dos clases de nodos en un clúster Storm: el nodo maestro (master node) y los nodos de trabajo (worker node). El nodo principal ejecuta un demonio (o servicio) llamado "Nimbus" similar al "JobTracker" de Hadoop. Nimbus es responsable de la distribución de código en todo el clúster, la asignación de tareas a las máquinas y el seguimiento de los fallos.

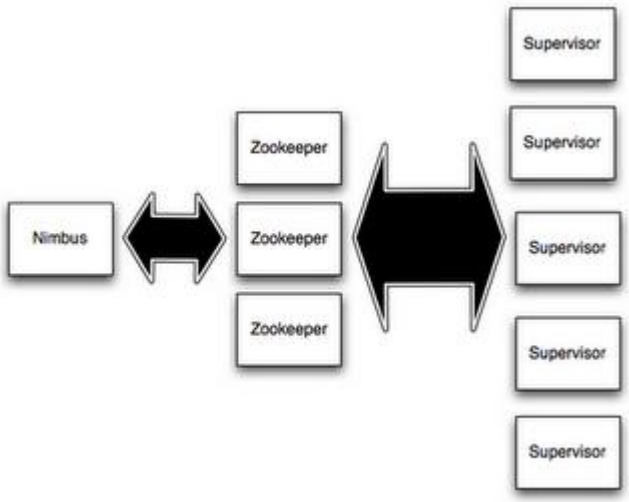


Figura 32: Componentes de un clúster Storm.

Cada nodo de trabajo (worker node) se ejecuta un servicio llamado "Supervisor". El supervisor supervisa el trabajo asignado a su máquina e inicia y detiene los procesos del worker cuando sea necesario, sobre la base de lo que el nodo Nimbus ha asignado. Cada proceso worker realiza la ejecución de un subconjunto de una topología. Una topología completa consiste en muchos procesos worker repartidos en múltiples máquinas.

Toda la coordinación entre Nimbus y los supervisores se realiza a través de un clúster Zookeeper (servicio centralizado para el mantenimiento de información relativa a configuración). Además, el demonio de Nimbus y el de Supervisor no guardan ningún estado. Todo el estado se mantiene en Zookeeper o en el disco local. Esto significa que se pueden parar el Nimbus o los supervisores y se iniciará automáticamente una copia de seguridad. Este diseño provoca que Storm sea enormemente estable. La relación entre estos tres nodos se muestra en la figura 32.

Topologías.

Para realizar los cálculos en tiempo real, se crea lo que se denominan "topologías" que no son más que gráficos de computación (ver Figura 33). Cada nodo en una topología contiene la lógica de procesamiento y los enlaces entre los nodos indican cómo se deben transportar los datos entre los nodos colindantes.

La ejecución de una topología es sencilla. En primer lugar, se debe empaquetar todo el código y las dependencias en un solo jar (archivo ejecutable JAVA). A continuación, se ejecuta un comando como el siguiente:

```
storm jar todo-mi-code.jar backtype.storm.MyTopology arg1 arg2
```

Esto ejecuta la clase backtype.storm.MyTopology con los argumentos arg1 y arg2. La función principal de la clase es definir la topología y presentarla al servicio Nimbus. El Storm jar se encarga de conectar con el servicio Nimbus y subir el jar.

En el siguiente subapartado se indaga más en la explicación de topologías.

Streams.

El núcleo de la abstracción en Storm es el "stream" (o flujo). Un stream es una secuencia ilimitada de tuplas. Storm proporciona los principios para la transformación de un stream en un nuevo stream de una manera distribuida y fiable. Por ejemplo, en relación con la red social Twitter, es posible transformar un stream de tweets en un stream de trending topics.

Los componentes básicos de Storm permiten realizar transformaciones en los streams son los "spouts" y "bolts". Estos elementos tienen interfaces que son implementadas para hacer funcionar la lógica específica de la aplicación.

Un spout es una fuente de stream. Por ejemplo, un spout puede conectarse a la API de Twitter y emitir una corriente de tweets. Suele ser el punto de comienzo de una topología.

Un bolt consume cualquier cantidad de stream de entrada, realiza algún tipo de procesamiento, y puede emitir nuevos streams. Para la transformación de streams complejos, como el cálculo de un stream de trending topics de una corriente de tweets, requieren múltiples pasos y, por tanto, múltiples bolts. Los bolts pueden hacer cualquier tipo de procesado, desde las funciones dirigidas, filtro de tuplas, efectuar agregaciones de streams o comunicarse con bases de datos.

Las redes de spouts y bolts son empaquetados en una "topología", que es el mayor nivel de abstracción en Storm. Como se ha comentado, una topología es un gráfico de computación que muestra las transformaciones que son efectuadas a streams y donde

cada nodo es un spout o bolt, como se representa en la figura 33.

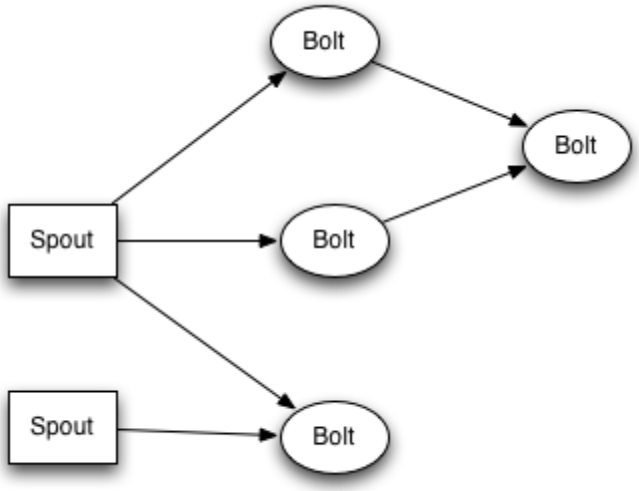


Figura 33: Ejemplo de una topología en Storm.

Cada nodo en una topología Storm se ejecuta en paralelo. En la topología, se puede especificar el nivel de paralelismo que se desea para cada nodo y, seguidamente, Storm generará el número de hilos especificado en el clúster para hacer la ejecución.

Una topología se ejecuta por siempre, o hasta que el usuario para el proceso. Si esto sucede, Storm reasignará automáticamente las tareas fallidas. Además, Storm garantiza que no habrá pérdida de datos, incluso si las máquinas se apagan y los mensajes se eliminan.

Modelo de datos.

Storm usa tuplas como modelo de datos. Una tupla es una lista de valores con un nombre y un campo de una tupla que puede ser un objeto de cualquier tipo. Storm es compatible con todos los tipos primitivos, Strings o matrices de bytes como valores de campo de tupla. Para utilizar un objeto de otro tipo, sólo tiene que implementar un serializador para el tipo.

Cada nodo en una topología debe declarar los campos de salida para las tuplas que emite.

2.7. ¿Por qué se ha elegido la opción propuesta?

A lo largo del capítulo dos se ha realizado un recorrido por una serie de herramientas alternativas a la tecnología escogida: Hadoop. Esta plataforma se ha elegido para realizar el trabajo fin de grado por diferentes razones:

- Funcionalidad: Sin asomo de duda, Hadoop es actualmente la herramienta de Big Data o análisis de datos que proporciona mayor funcionalidad. Como se ha visto durante el capitulo 2, muchas de las herramientas alternativas solo cumplen una de las dos funcionalidades principales que Hadoop ofrece. El resto de herramientas, a pesar de cubrir las dos posibilidades, no presentan el mismo nivel de desarrollo y se encuentran en versiones iniciales. Además no presentan tan buenas características como Hadoop a la hora de analizar grandes ficheros de datos ya que el HDFS está meditado para tratar con grandes volúmenes de datos.

- Escalabilidad: Hadoop es una plataforma de almacenamiento de datos con una alta escalabilidad porque es capaz de guardar y distribuir grandes conjuntos de datos entre cientos de computadores convencionales que operan en paralelo al contrario que las bases de datos tradicionales. Además es muy sencillo aumentar la capacidad del sistema, tan solo hace falta añadir más computadores al clúster para tener mayor capacidad de procesamiento. Esto nos permite ejecutar aplicaciones en cientos de nodos que involucran cientos de terabytes de datos.

- Solución gratuita: Al contrario que algunas herramientas para el análisis de datos (como SAS o SPSS), Hadoop es gratuito. Asimismo la construcción de un clúster Hadoop se realiza con computadoras convencionales de poco precio, por tanto, se trata de una solución rentable.

- Flexibilidad: Hadoop permite fácil acceso a nuevas fuentes de datos así como aprovechar tanto archivos estructurados como desestructurados. Esto lo hace idóneo para análisis de archivos procedentes de redes sociales. Además al tratarse de una plataforma libre existe una gran comunidad de desarrolladores que permite a Hadoop mantenerse actualizado para analizar nuevas fuentes de datos.

- Creciente desarrollo: Como se ha comentado, Hadoop es una plataforma libre lo que permite que cualquier persona pueda colaborar en su desarrollo. Aparte de la comunidad de desarrolladores existen numerosas empresas privadas dedicadas al desarrollo de Hadoop, entre las que destacan Hortonworks y Cloudera. Esto asegura que Hadoop se mantenga en constante evolución y garantiza su uso en los próximos años. Todas estas fuentes de colaboración han permitido la creación de numerosas herramientas y mejoras dentro del entorno Hadoop que maximizan su rendimiento y facilitan su utilización para el usuario.

- Base JAVA: Hadoop está desarrollado en tecnología JAVA lo cual permite desarrollar programas para la plataforma en este lenguaje. Es también este hecho un motivo para elegir Hadoop: aprovechar el conocimiento previo en JAVA. A pesar de que este no será el lenguaje de programación principal del presente trabajo.

Es evidente el creciente interés que despierta en la actualidad todo lo relacionado con Big Data, y es presumible que será una disciplina muy importante en el ámbito empresarial en los próximos años. Dentro de este marco Hadoop supone una revolución ya que proporciona un nuevo acercamiento al análisis de datos y a su almacenamiento. Es, por tanto, el futuro uso de este tipo de plataformas una razón de peso decantarse por su estudio y aprendizaje.

Capítulo 3. Tecnología utilizada.

Siguiendo con el índice descrito en el capítulo 1, se va a comenzar a exponer el tercer capítulo. En este apartado se va a desarrollar con detalle todos los aspectos relativos a la herramienta utilizada durante el trabajo: Hadoop. Esta sección ayudará a comprender todos los aspectos teóricos de la plataforma, su funcionamiento y se realizarán dos ejemplos que ayudarán a entender mejor la tecnología de forma práctica.

3.1. Big Data

Como se comentó en el primer capítulo el concepto de Big Data se refiere al almacenamiento y procesado de enormes cantidades de datos, tan desproporcionadamente grandes que resulta imposible tratarlos con las herramientas de bases de datos convencionales. Sin embargo, Big Data no se refiere a alguna cantidad en específico, ya que es usualmente utilizado cuando se habla en términos de petabytes y exabytes de datos. Lo que para una determinada empresa es Big Data puede no serlo para otra compañía, ello depende de los recursos de los que cada una disponga (ver Figura 34).

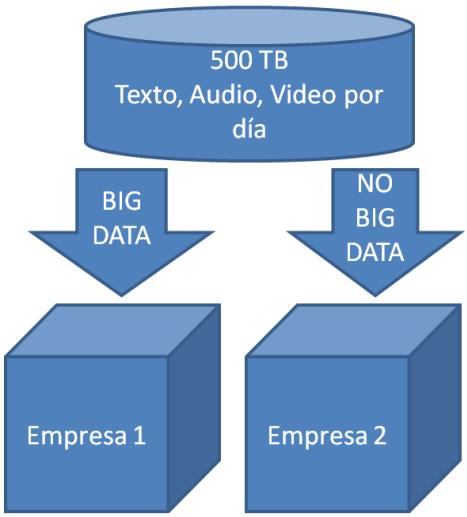


Figura 34: Big Data en empresas.

Se trata de miles de millones de registros que deben manejar internamente algunas empresas para tratar la proliferación de páginas web (Google), petabytes de imagen y vídeo (YouTube), dispositivos móviles y aplicaciones (Apple), sensores meteorológicos (NationalWeather) y muchas otras fuentes de datos. Esto se refiere a empresas y organizaciones capaces de generar más de 2.5 quintillones de bytes al día, hasta el punto de que el 90% de los datos del mundo se han creado durante los últimos dos años, como muestra el crecimiento de la figura 35. Una curva claramente exponencial. Pero existen muchas otras fuentes de datos, por ejemplo, un Air Bus genera 10 terabytes de datos cada 30 minutos, mas de 2 billones de personas utilizan Internet diariamente y a finales de este año se prevé en 4.8 ZB el trafico total en internet, Twitter genera 12 TB de datos diariamente con más de 200 millones de usuarios activo, la bolsa de Nueva York (NYSE) genera 1 TB de datos diarios y, como estos, numerosos ejemplos más.

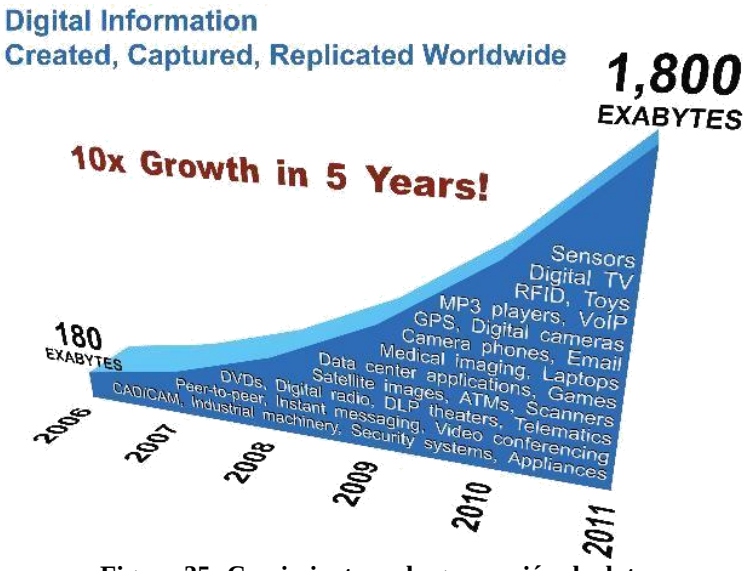


Figura 35: Crecimiento en la generación de datos.

Además del gran *volumen* de información, esta existe en una gran *variedad* de datos que pueden ser representados de diversas maneras en todo el mundo, por ejemplo de dispositivos móviles, audio, video, sistemas GPS, incontables sensores digitales en equipos industriales, automóviles, medidores eléctricos, veletas, anemómetros, etc. Los cuales pueden medir y comunicar el posicionamiento, movimiento, vibración, temperatura, humedad y hasta los cambios químicos que sufre el aire, de tal forma que las aplicaciones que analizan estos datos requieren que la *velocidad* de respuesta sea lo demasiado rápida para lograr obtener la información correcta en el momento preciso.

Tipos de datos.

Los principales tipos de datos se resumen en 5 y se recogen en la figura 36:

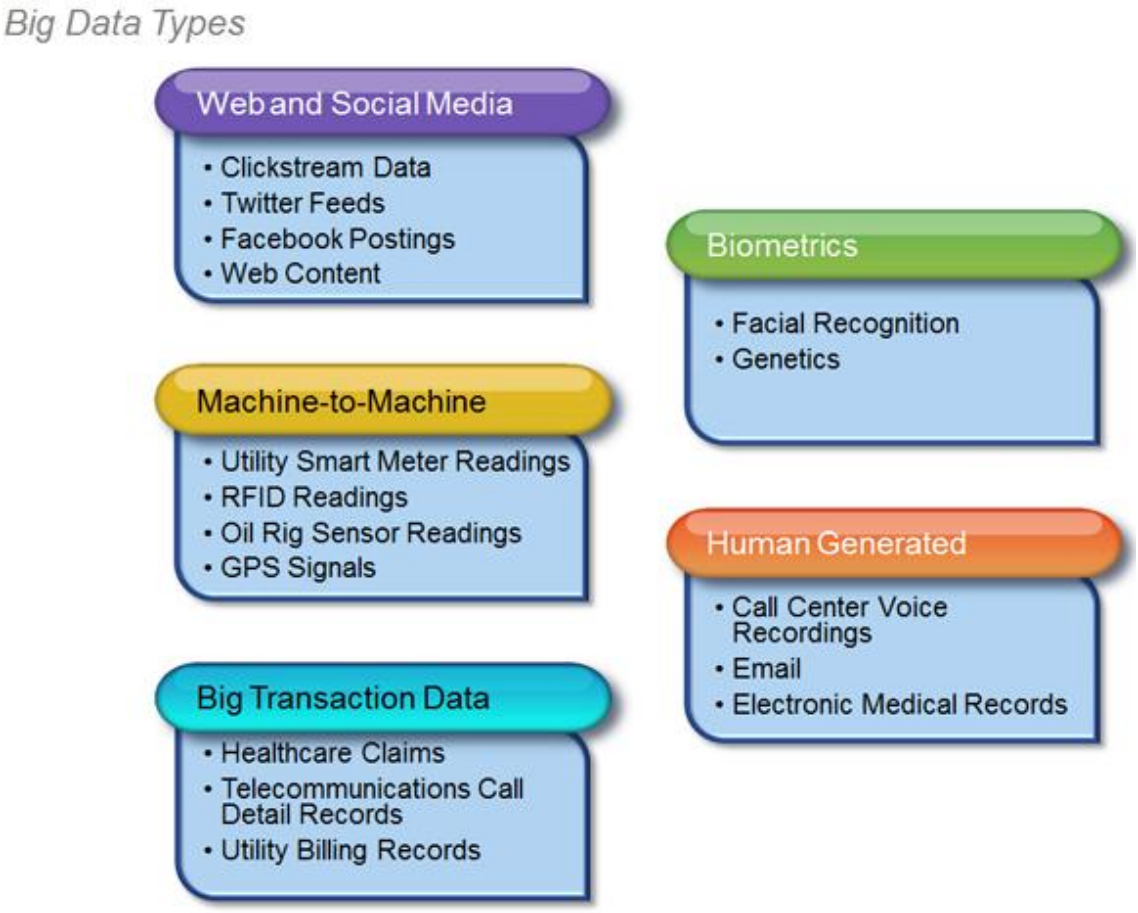


Figura 36: Tipos de fuentes de datos.

- Web and Social Media: Incluye contenido web e información que es obtenida de las redes sociales como Facebook, Twitter, LinkedIn o blogs.
- Machine-to-Machine (M2M): M2M se refiere a las tecnologías que permiten conectarse a otros dispositivos. M2M utiliza dispositivos como sensores o medidores que capturan algún evento en particular (velocidad, temperatura,

presión, variables meteorológicas o variables químicas como la salinidad) los cuales transmiten a través de redes alámbricas, inalámbricas o híbridas a otras aplicaciones que traducen estos eventos en información significativa.

- Big Transaction Data: Incluye registros de facturación, en telecomunicaciones registros detallados de las llamadas (CDR), etc. Estos datos transaccionales están disponibles en formatos tanto semiestructurados como no estructurados.

- Biometrics: Información biométrica en la que se incluye huellas digitales, escaneo de retina, reconocimiento facial, genética, etc. En el área de seguridad e inteligencia, los datos biométricos han sido información importante para las agencias de investigación.

- Human Generated: Las personas generamos diversas cantidades de datos como la información que guarda un call center al establecer una llamada telefónica, notas de voz, correos electrónicos, documentos electrónicos, estudios médicos, etc.

Funciones del Data Scientist o "científico de los datos" (analista de datos).

Las funciones del Data Scientist o “científico de los datos” se engloban en 3 fundamentalmente:

- Data wranling: Captura y almacenamiento de la información. Es el procedimiento manual de convertir “raw data” (información en bruto) en información con formato para que pueda ser analizada. Suele ocupar el 80% del tiempo de trabajo del data Scientist.

- Data analysis: Obtención de valor a partir de la información.

- Data visualization: Visualizado de los resultados.

El flujo de trabajo habitual de un data Scientist se recoge en la figura 37.

Figura 37: Esquema de funciones del Data Scientist.

3.2. Apache Hadoop.

Como se comentó en el primer capítulo, Apache Hadoop es una plataforma que permite el procesamiento de grandes volúmenes de datos a través de clúster, usando un modelo simple de programación. Proporciona un framework, escrito en Java, sobre el cual desarrollar aplicaciones distribuidas que requieren un uso intensivo de datos y de alta escalabilidad. Este proyecto es administrado por Apache Software Foundation.

Se presenta como una solución para los programadores sin experiencia en desarrollo de aplicaciones para entornos distribuidos, dado que oculta la implementación de detalles propios de estos sistemas: paralelización de tareas, administración de procesos, balanceo de carga y tolerancia a fallos.

Muchos colaboradores de Hadoop trabajan en algunas de las compañías más grandes del mundo de la tecnología. Originalmente fue desarrollado y empleado por las grandes empresas dominantes en la Web, como Yahoo y Facebook. Ahora Hadoop es muy utilizado en finanzas, tecnología, telecomunicaciones, medios y entretenimiento, gobierno, instituciones de investigación y otros mercados con gran cantidad de datos. Con Hadoop, las empresas pueden explorar datos complejos mediante el análisis personalizado adaptado a sus datos y necesidades.

Hadoop fue diseñado para analizar de forma rápida y fiable los datos estructurados y complejos. Como resultado, muchas empresas han optado por desplegar Hadoop junto a sus demás sistemas informáticos, lo que les permite combinar los datos antiguos y los nuevos de distintas formas novedosas.

Hadoop es muy útil cuando se van a realizar proyectos que necesiten de escalabilidad. Al disponer los datos de forma distribuida, la búsqueda se puede realizar muy

rápidamente ya que Hadoop puede acceder a ella de forma paralela. Y aunque los datos estén distribuidos, no hay que preocuparse de fallos ya que dispone de un sistema de seguridad.

Esta tecnología fue creada por Doug Cutting, el creador de Apache Lucene, la biblioteca de búsqueda de texto ampliamente utilizada. Hadoop tiene sus orígenes en Apache Nutch, un motor de búsqueda de código abierto, que es en sí una parte del proyecto Lucene.

En 2004, Google publicó el paper (o artículo) que introdujo MapReduce al mundo. A principios de 2005, los desarrolladores de Nutch tenían una implementación de MapReduce en Nutch, ya mediados de ese año todos los principales algoritmos Nutch habían sido portados para ser ejecutados utilizando MapReduce y NDFS (sistema de archivos de Nutch).

NDFS y la implementación de MapReduce en Nutch eran aplicables más allá del ámbito de la búsqueda, y en febrero de 2006 se trasladaron fuera de Nutch para formar un subproyecto independiente de Lucene llamado Hadoop. Al mismo tiempo, Doug Cutting se unió a Yahoo!, lo que proporcionó a Hadoop un equipo especializado y los recursos necesarios para convertirlo en un sistema de éxito.

En enero de 2008, Hadoop formó su propio proyecto de nivel superior en Apache, lo que confirma su éxito y su comunidad diversa y activa. Por entonces, Hadoop estaba siendo utilizado por muchas otras empresas, además de Yahoo!, como Last.fm, Facebook o el New York Times.

Arquitectura (Hadoop 1.0).

La arquitectura de Hadoop hasta la versión 2.0, cuyos componentes principales se muestran en la figura 38, se vertebra sobre tres pilares fundamentales:

- Sistema de ficheros: Hadoop se apoya para su funcionamiento en un sistema de ficheros distribuido, denominado HDFS.

- Hadoop MapReduce: El motor de Hadoop consta de un planificador de trabajos MapReduce, así como de una serie de nodos encargados de llevarlos a cabo.

- Hadoop Common: Conjunto de utilidades que posibilitan la integración de subproyectos de Hadoop.

Sobre el sistema de ficheros se ubica el motor de MapReduce, que consiste en un planificador de trabajos denominado JobTracker, a través del cual las aplicaciones cliente envían trabajos MapReduce. Este planificador envía el flujo de trabajo entrante a los nodos TaskTracker disponibles en el clúster, que se ocuparán de ejecutar las funciones map y reduce en cada nodo. El planificador trata de mantener esos trabajos tan cerca de la máquina que ha emitido esa información como sea posible. Si el trabajo

no puede ser ubicado en el nodo actual en el que la información reside, se le da prioridad a nodos en el mismo rack. Esto reduce el tráfico de red en la red principal del clúster. Si un TaskTracker falla o sufre un timeout, esa parte del trabajo se re planifica. Notar que Hadoop responde a una estructura master/slave (maestro/esclavo) donde el JobTracker está situado en el maestro mientras que existe un TaskTracker por cada máquina esclavo. Sobre estos términos se seguirá hablando en el siguiente subapartado.

Por otra parte, el JobTracker registra los trabajos pendientes de ejecución que residan en el sistema de ficheros. Cuando arranca un JobTracker busca esa información, de tal forma que pueda volver a empezar el trabajo a partir del punto en el que se quedó.

Respecto al sistema de ficheros de Hadoop (HDFS), se tienen dos elementos fundamentales en la arquitectura: el NameNode y los DataNode. El NameNode se encuentra únicamente en el nodo máster y se encarga de mantener indexados todos los datos almacenados. Es decir, le dice a la aplicación donde se ubican los datos que busca. Los NameNode se encuentran en las computadoras slaves y se encargan de almacenar los datos.

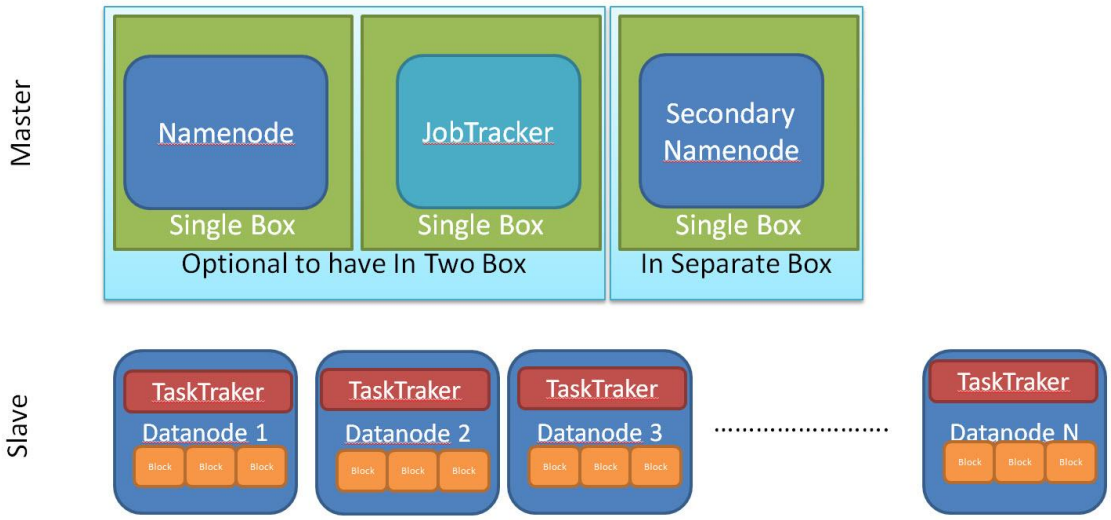


Figura 38: Arquitectura básica de Hadoop MapReduce.

Tanto los componentes de HDFS como los de MapReduce para Hadoop, serán explicados detalladamente en los siguientes apartados con la intención de entender completamente el funcionamiento de la tecnología Hadoop. Se ha reservado un subapartado para estos dos elementos.

Sería importarte señalar que esta es la arquitectura principal con la que Apache Hadoop surgió. A partir de la versión Hadoop 2.0 veremos como esta arquitectura cambia ligeramente dando paso a otro componente fundamental: YARN. Esta circunstancia se explicará en siguientes apartados. La razón por la cual se explica la estructura de inicio es que es la utilizada en la versión Hadoop 1.2.1, la última versión

estable del software y aún muy utilizada.

Modos de ejecución.

Hadoop se puede ejecutar de tres modos distintos:

- Modo local / standalone: Por defecto, Hadoop está configurado para ejecutarse en modo no-distribuido como un proceso Java aislado. Esto es útil para depuración.

- Modo pseudo-distribuido: Hadoop también puede ejecutarse en un único modo en un modo pseudo-distribuido donde cada demonio Hadoop se ejecuta en un proceso Java diferente.

- Modo distribuido: Esta es la forma de aprovechar toda la potencia de Hadoop, ya que se maximiza el paralelismo de procesos y se utilizan todos los recursos disponibles del clúster en el que se va a configurar Hadoop.

3.2.1. MapReduce.

Paradigma MapReduce.

Como se ha comentado al comienzo de este apartado, el modelo de programación MapReduce fue introducido por primera vez por Google a finales del año 2004. El objetivo era crear un framework adaptado a la computación paralela que permitiera procesar y generar grandes colecciones de datos sobre máquinas genéricas, sin la necesidad de utilizar supercomputadores o servidores dedicados, y que fuera fácilmente escalable.

En esencia, el modelo que propone MapReduce es bastante sencillo. El programador debe encargarse de implementar dos funciones, map y reduce, que serán aplicadas a todos los datos de entrada. Tareas como hacer particiones de los datos de entrada, despliegue de maestro y trabajadores, esquema de asignación de trabajos a los trabajadores, comunicación y sincronización entre procesos, tratamiento de caídas de procesos, quedan a cargo del entorno de ejecución, liberando de esa manera al programador. En la figura 39, se muestra un esquema muy general de las tareas principales del paradigma MapReduce.

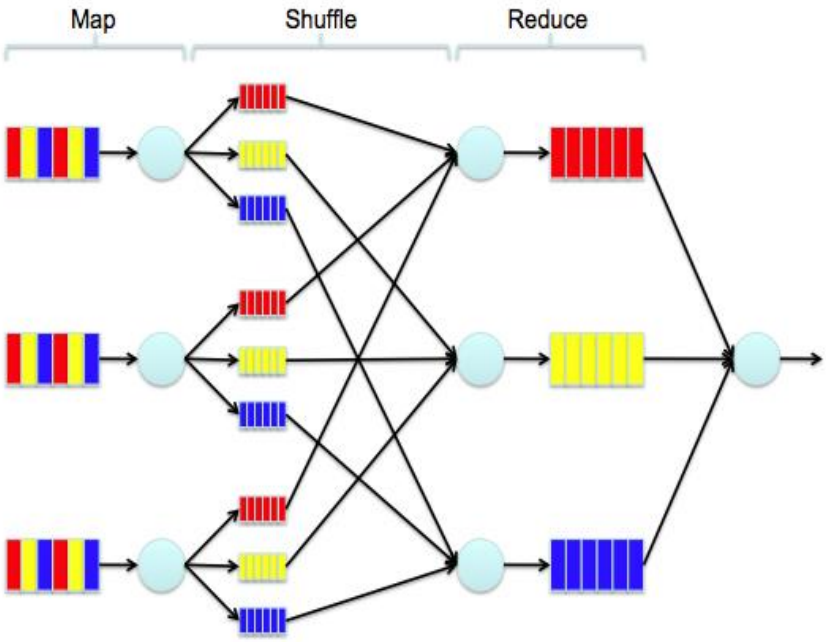


Figura 39: Esquema funcionamiento MapReduce.

Las funciones map y reduce están definidas ambas con respecto a datos estructurados en parejas <clave, valor>. La función map es aplicada en paralelo sobre cada pareja de entrada (k1,v1), generando una lista intermedia de parejas <clave, valor> (lista(k2,v2)). La función reduce, es aplicada en paralelo sobre cada grupo de parejas con la misma clave intermedia (k2), tomando los valores (v2) contenidos en el grupo para generar como salida una nueva lista de parejas <clave, valor> (lista(k3,v3)). Entre las fases map y reduce, existe una operación de agrupación que reúne todos los pares con la misma clave de todas las listas, creando un grupo por cada una de las diferentes claves generadas. Las tuplas generadas por la función reduce, son grabadas de forma permanente en el sistema de ficheros distribuido. Por tanto, la ejecución de la aplicación acaba produciendo r ficheros, donde r es el número de tareas reduce ejecutadas. La Figura 40 muestra una visión simplificada del paradigma de programación MapReduce.

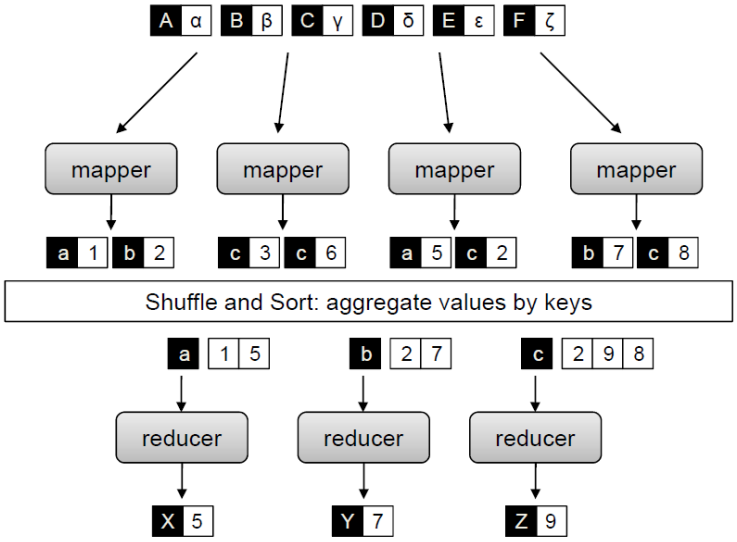


Figura 40: Esquema simplificado MapReduce.

Una de las ideas más importantes que hay detrás de MapReduce, es separar el qué se distribuye del cómo. Un trabajo MapReduce está formado por el código asociado a las funciones map y reduce, junto con parámetros de configuración. El desarrollador envía el trabajo al planificador y el entorno de ejecución se encarga de todo lo demás, gestionando el resto de aspectos relacionados con la ejecución distribuida del código, como son:

- Planificación: Cada aplicación MapReduce es dividida en unidades más pequeñas denominadas tareas (tasks). Dado que el número de tareas puede ser superior al de los nodos del clúster, es necesario que el planificador mantenga una cola de tareas y vaya haciendo un seguimiento del progreso de las tareas en ejecución, asignando tareas de la cola de espera a medida que los nodos van quedando disponibles.

- Co-ubicación datos/código: Una de las ideas clave de MapReduce es mover el código y no los datos. Esta idea está ligada con la planificación y depende, en gran medida, del diseño del sistema de ficheros distribuido. Para lograr la localidad de datos, el planificador intentará ejecutar la tarea en el nodo que contiene un determinado bloque de datos necesario para la tarea.

- Sincronización: En MapReduce, la sincronización se consigue mediante una “barrera”. Una “barrera”, es un mecanismo de sincronización entre procesos que espera a que todos los procesos de un lado de la barrera terminen antes que empiecen los procesos del otro lado. Esto significa que la fase map debe terminar antes que empiece la fase reduce.

- Manejo de errores y fallos: El entorno de ejecución debe cumplir con todas las tareas mencionadas en los puntos anteriores, en un entorno donde los errores y los fallos son la norma. MapReduce fue diseñado expresamente para ser ejecutados en servidores de gama baja, por lo que el entorno de ejecución debe ser especialmente resistente. En grandes clústeres de máquinas, los errores de disco y de RAM son comunes.

Anteriormente, se ha presentado una visión simplificada de MapReduce. Existen dos elementos adicionales que completan el modelo de programación, denominados combiner y partitioner. La Figura 41 muestra una visión más completa de MapReduce.

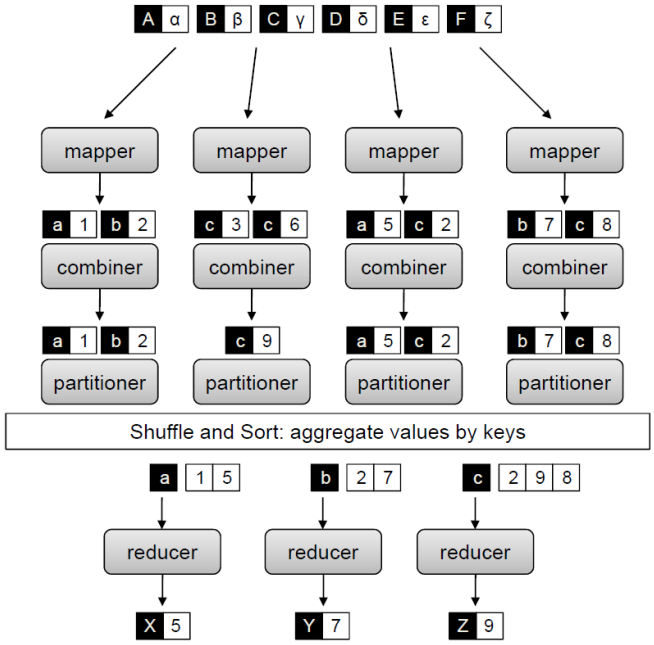


Figura 41: Esquema completo MapReduce.

El partitioner es el responsable de dividir el espacio de claves intermedias y asignar parejas <clave, valor> intermedias a las tareas reduce. Dicho de otro modo, el partitioner especifica la tarea a la cual debe asignarse una pareja <clave, valor>. Para ello, calcula el valor hash de la clave (la transforma en un valor alfanumérico) y obtiene el módulo del valor en base al número de tareas reduce a ejecutar. El objetivo es asignar el mismo número de claves a cada reduce. Sin embargo, dado que el partitioner sólo tiene en cuenta la clave y no los valores, puede haber grandes diferencias en el volumen de tuplas <clave, valor> enviadas a cada reduce, dado que diferentes claves pueden tener diferente número de valores asociados.

El combiner representa un punto de optimización en el modelo MapReduce. En algunos casos, existe una repetición significativa en las claves intermedias producidas por cada map. Todas esas repeticiones deberían ser enviadas, a través de la red, al nodo asignado para procesar una tarea reduce para una clave dada. Este modelo se

muestra del todo ineficiente, por lo que el desarrollador tiene la opción de definir una función combiner. Esta función le permite fusionar, parcialmente, los datos antes de ser enviados a través de la red. La función combiner se ejecuta en cada nodo donde se procesa una tarea map. Típicamente, se utiliza el mismo código para implementar la función combiner y la función reduce. La diferencia radica en el modo en que MapReduce gestiona la salida de la función. La salida de la función reduce se escribe en el sistema de ficheros de forma permanente. La salida de la función combiner se escribe en un fichero intermedio que será enviado a una tarea reduce.

Hadoop MapReduce.

Este subapartado tiene como finalidad profundizar en todos los conceptos introducidos en el apartado de arquitectura y en el del paradigma de programación MapReduce [Ref8].

Hadoop, como ya se ha ido describiendo, proporciona un entorno de ejecución orientado a aplicaciones desarrolladas bajo el paradigma de programación MapReduce. Bajo este modelo, la ejecución de una aplicación presenta dos etapas coincidentes con las descritas anteriormente:

- Map: donde se realiza la ingestión y la transformación de los datos de entrada, en la cual los registros de entrada pueden ser son procesados en paralelo.

- Reduce: fase de agregación o resumen, donde todos los registros asociados entre sí deben ser procesados juntos por una misma entidad.

La idea principal sobre la cual gira el entorno de ejecución Hadoop MapReduce es que la entrada puede ser dividida en fragmentos y, cada fragmento, puede ser tratado de forma independiente por una tarea map. Los resultados de procesar cada fragmento, pueden ser físicamente divididos en grupos distintos. Cada grupo se ordena y se pasa a una tarea reduce.

Una tarea map puede ejecutarse en cualquier nodo de cómputo del clúster, y múltiples tareas map pueden ejecutarse en paralelo en el clúster. La tarea map es responsable de transformar los registros de entrada en parejas <clave, valor>. La salida de todos los map se dividirá en particiones, y cada partición será ordenada por clave. Habrá una partición por cada tarea reduce. Tanto la clave como los valores asociados a la clave son procesados por la tarea reduce, siendo posible que varias tareas reduce se ejecuten en paralelo.

En la Figura 42, se muestra de forma esquemática el ciclo de ejecución de una aplicación en Hadoop. El desarrollador únicamente proporciona al framework Hadoop cuatro funciones: la función que lee los registros de entrada y los transforma en tuplas <clave, valor> (RecordReader), la función map (Mapper), la función reduce (Reducer), y la función que transforma las parejas <clave, valor> generadas por la función reduce en registros de salida (RecordWriter).

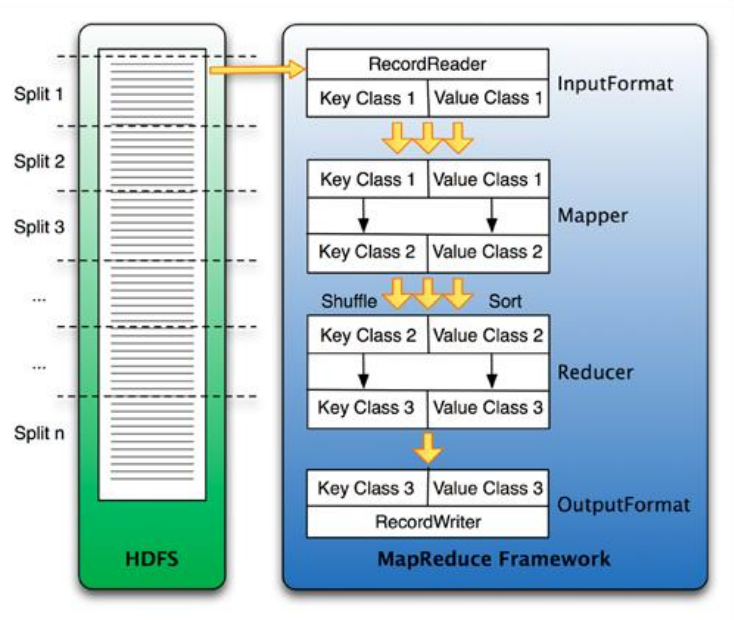


Figura 42: Ciclo de aplicación Hadoop.

El entorno de ejecución Hadoop Mapreduce, como ya se ha comentado, está formado por dos componentes principales: JobTracker y TaskTracker, ambos codificados en Java. Al igual que en el HDFS, el entorno de ejecución presenta una arquitectura cliente/servidor o master/slave (maestro/esclavo). En un clúster hay un único JobTracker, siendo su labor principal la gestión de los TaskTrackers, entre los que distribuye los trabajos MapReduce que recibe. Los TaskTrackers son los encargados de ejecutar las tareas map/reduce. En un clúster típico, se ejecuta un TaskTracker por nodo de cómputo. En la Figura 43 se muestra de forma esquemática la interacción entre JobTracker y TaskTracker.

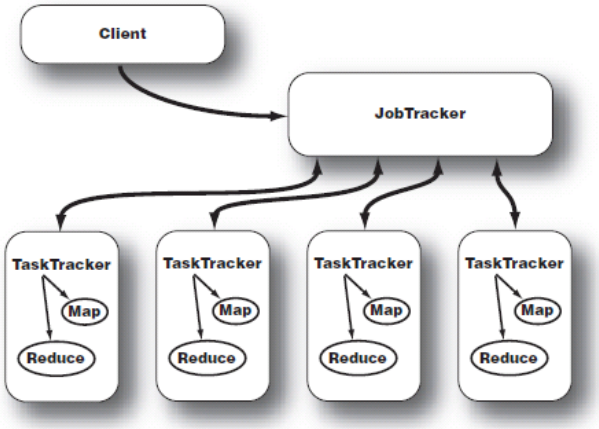


Figura 43: Interacción JobTracker-TaskTracker.

El JobTracker es el enlace entre la aplicación y Hadoop. Una vez se envía un trabajo al clúster, el JobTracker determina el plan de ejecución en base a los ficheros a procesar, asigna nodos de cómputo a las diferentes tareas, y supervisa todas las tareas que se están ejecutando. En el caso de fallar una tarea, el JobTracker relanza la tarea, posiblemente en un nodo diferente, existiendo un límite predefinido de intentos en el caso que dicha tarea falle de forma reiterada. Como se ha comentado anteriormente, sólo se ejecuta un JobTracker por clúster Hadoop.

Cada TaskTracker es responsable de ejecutar las tareas que el JobTracker le ha asignado. Cada TaskTracker puede ejecutar varias tareas map/reduce en paralelo. El JobTracker crea instancias Java separadas para la ejecución de cada tarea. De esa manera se garantiza que, en caso de fallar la ejecución de una tarea, no afecta al resto de tareas ni tampoco al propio JobTracker. En la Figura 44 se muestra el esquema de ejecución de aplicaciones en un entorno Hadoop.

El TaskTracker se comunica con el JobTracker a través de un protocolo de latidos (heart beat protocol). En esencia, el latido es un mecanismo que utiliza el TaskTracker para anunciar su disponibilidad en el clúster. El protocolo de latido es lo que permite saber al JobTracker que el TaskTracker está disponible o en funcionamiento. Además de anunciar su disponibilidad, el protocolo de latidos también incluye información sobre el estado del TaskTracker. Los mensajes de latido indican si el TaskTracker está listo para la siguiente tarea o no. Cuando el JobTracker recibe un mensaje de latido de un TaskTracker, declarando que está listo para la siguiente tarea, el JobTracker selecciona el siguiente trabajo disponible en la lista de prioridades y determine qué tarea es la más apropiada para el TaskTracker.

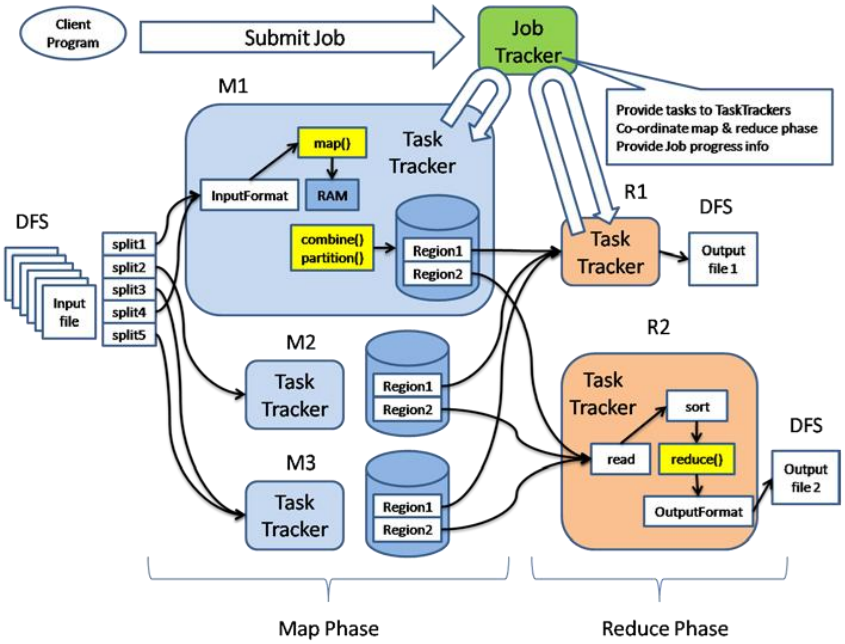


Figura 44: Esquema de ejecución.

Los TaskTrackers están limitados por el número de tareas que pueden ejecutar. Por ejemplo, un TaskTracker puede ser configurado para procesar dos tareas map y dos tareas reduce de forma paralela en un momento dado. Por esta razón, el JobTracker tiene que averiguar qué tipo de tarea debe asignar al TaskTracker. Si tiene disponibilidad para ejecutar tarea map, se le asigna una tarea map, de lo contrario se le asigna una tarea reduce. El número de tareas que un TaskTracker puede ejecutar al mismo tiempo, depende de la cantidad de núcleos y de la memoria disponible en el nodo donde se está ejecutando.

Hadoop trata de ser eficiente en el tratamiento de las tareas, considerando la localidad de los datos de la tarea a procesar. A cada tarea map de un trabajo Hadoop, se le asigna una parte de los datos de entrada, pudiendo estar ubicados dichos datos en cualquier parte del HDFS del clúster. El planificador, inicialmente intentará que una tarea se ejecute en el nodo que contienen los datos a nivel local. Cuando los datos a procesar están almacenados de manera local en el nodo donde se ejecuta la tarea, el TaskTracker se libera de tener que descargar los datos necesarios de un nodo remoto. La localidad de los datos asegura un mejor rendimiento y eficiencia. Cuando una tarea no puede ser asignada a un nodo que contiene los datos a nivel local, Hadoop intentará asignar la tarea al nodo más cercano a los datos. En este contexto, el nodo más cercano sería un nodo en el mismo rack donde se almacenan los datos. Por último, si no se puede asignar la tarea a un nodo del mismo rack, entonces la alternativa es encontrar un nodo en otro rack.

El número de tareas map a ejecutar depende de varios factores: el número de tareas map especificado por el programador sirve como referencia para el entorno de ejecución, pero el número de tareas que finalmente ejecutará tendrá en cuenta tanto el número de ficheros a procesar como el número de bloques HDFS que ocupan dichos ficheros. En cambio, el número de tareas reduce es igual al número de tareas especificado por el programador.

El JobTracker mantiene una cola de trabajos enviados para su ejecución en el clúster y realiza un seguimiento del progreso de cada trabajo. Los clientes pueden configurar los trabajos en Hadoop con un nivel de prioridad que representa la importancia del trabajo respecto a otros trabajos en la cola.

Por defecto, FIFO (First In, First Out) es la política de planificación que se utiliza en Hadoop para priorizar los trabajos de la cola. Dado que este tipo de planificación puede no ser muy eficiente en entornos productivos, el planificador se sacó fuera de núcleo para, de esta manera, permitir el uso de un planificador alternativo en el caso que fuera necesario. Dos de los más extendidos son Fair Scheduler y Capacity Scheduler:

- Fair Scheduler fue desarrollado por Facebook. El objetivo del planificador es, proporcionar una rápida respuesta a trabajos pequeños y calidad de servicio (QoS) para trabajos de producción. Se basa en tres conceptos básicos: los trabajos se agrupan en pools, cada pool tiene asignada una porción del clúster mínima garantizada y el exceso de capacidad se distribuyen entre los trabajos. Los trabajos que están sin categorizar van a un pool por defecto.

- Capacity Scheduler fue desarrollado por Yahoo y presenta algunas similitudes con Fair Scheduler. Existen una serie de colas al estilo de Fair Scheduler, que pueden estar organizadas de forma jerárquica y cada cola tiene una capacidad asignada. Cada cola está planificada siguiendo una política FIFO con prioridades. Este tipo de planificador permite a los usuarios simular un clúster MapReduce separado, con una planificación FIFO en cada uno.

En la Figura 45 se muestra el esquema detallado de ejecución de una aplicación desarrollada bajo el paradigma MapReduce. Como se puede observar, además de las fases map y reduce, existen otras fases a tener en cuenta a la hora de ejecutar una aplicación en Hadoop.

A continuación, se describe cada una de las fases que presenta cada proceso. Ya que la mayoría de fases son funciones internas del propio framework de ejecución, tienen asociados una serie de parámetros de configuración que pueden afectar al rendimiento del propio clúster. La configuración de dichos parámetros puede ser realizada por el propio programador o por el administrador del sistema.

Realizar la sintonización de los parámetros de configuración no es una tarea sencilla, dado que Hadoop presenta más cientos de parámetros ajustables. Además, un único parámetro puede tener efectos importantes en el rendimiento global del sistema. Por ese motivo, en la descripción que se realiza de cada una de las fases, se indican los parámetros asociados. De esta manera es posible formarse una idea de la dificultad que entraña la sintonización del sistema.

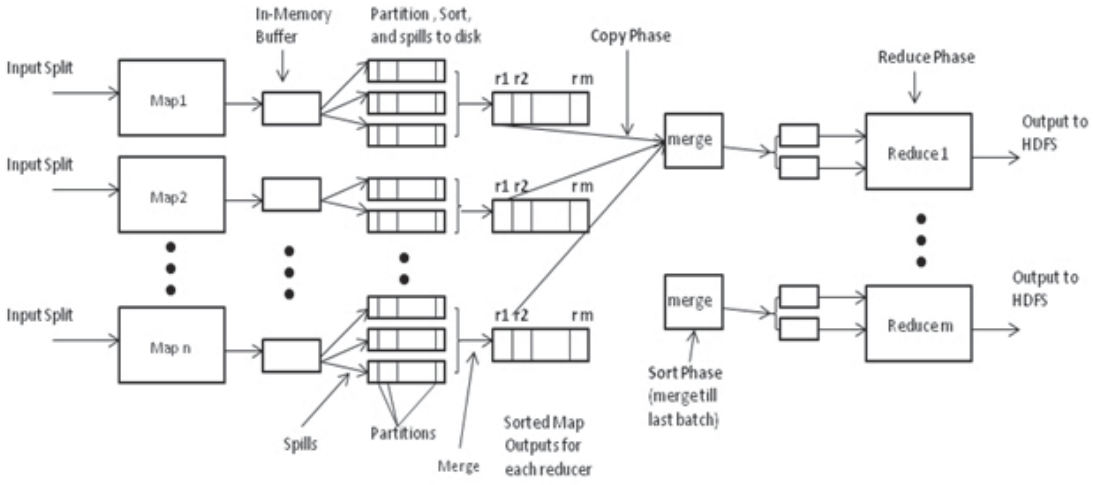


Figura 45: Esquema detallado de ejecución.

Tarea map. Presenta las siguientes fases:

- Procesamiento map: a cada tarea map(Map 1, Map 2, ..., Map n) se le asigna un bloque del fichero (Input Split). Cada bloque es dividido en parejas del tipo

<clave, valor>. La función map es invocada para cada pareja. La información generada por la función map es escrita en un buffer de memoria circular asociado con cada tarea map (In Memory Buffer). El tamaño del buffer viene fijado por la propiedad io.sort.mb, siendo el tamaño por defecto de 100Mb.

- Spill: cuando el buffer llega al porcentaje de ocupación fijado por la propiedad io.sort.spill.percent(por defecto 0,80 que corresponde a un 80%), un proceso ejecutado en segundo plano realiza el volcado del contenido del buffer a disco (Partitions). Mientras el volcado tiene lugar, la tarea map sigue escribiendo en el buffer mientras éste tenga espacio libre. El volcado se realiza siguiendo una política round-robinen un subdirectorio específico creado para el trabajo en ejecución. Este subdirectorio reside bajo el directorio especificado en la propiedad mapred.local.dir. Se crea un nuevo fichero cada vez que el buffer alcanza el porcentaje de ocupación fijado. A estos ficheros se les denomina ficheros de spill.

- Particionamiento: antes de escribir a disco, un proceso en segundo plano divide la información en tantas particiones (r1, r2,..., rm) como tareas reduce se han programado para el trabajo en curso.

- Sorting: se realiza en memoria una ordenación por clave de cada partición. En el caso que la aplicación hubiera definido una función combiner, ésta procesará la salida generada por la ordenación. El propósito de la función combiner es reducir el volumen de datos a entregar a la fase Reduce agrupando los registros que comparten la misma clave.

- Merge: antes de que la tarea map finalice, los diferentes ficheros de spill generados son intercalados generándose un fichero de salida ordenado. La propiedad io.sort.factorcontrola el máximo número de ficheros a fusionar a la vez, que por defecto es 10.

Tarea reduce: Los ficheros particionados generados en la fase map son proporcionados a las tareas reduce (Reduce 1,..., Reduce m) vía HTTP. El número de hilos usados para servir dichos ficheros viene fijado por la propiedad tasktracker.http.threadsy por defecto está fijado a 40 hilos para cada TaskTracker. Presenta las siguientes fases:

- Copy: Tan pronto como las tareas map finalizan la información correspondiente a cada tarea reduce es copiada. La fase reduce tiene un número de hilos que realizan dichas copias. Dicho número está fijado por la propiedad mapred.reduce.parallel.copies (por defecto 5). La información de salida que genera el map es copiada a un buffer del TaskTracker controlado por la propiedad mapred.job.shuffle.input.buffer.percent (por defecto 0,70 que corresponde al 70%) que indica la proporción de memoria heap utilizada para

tal propósito. Cuando dicho buffer llega al porcentaje de ocupación indicado por la propiedad mapred.job.shuffle.merge.percent (por defecto 0,66 o 66%), o se ha llegado al número máximo de ficheros de salida de tareas mapescritos en el búfer, fijado por la propiedad mapred.job.shuffle.merge.percent (por defecto 1.000), son fusionados y volcados a disco. Dado que las copias se acumulan en disco, un proceso en segundo plano los fusiona y ordena en ficheros más grandes ahorrando tiempo en subsiguientes fusiones.

- Merge: Cuando todos los ficheros de salida de las tareas map han sido copiados, son intercalados manteniendo la ordenación con la que se han generado en el proceso map. Este proceso se realiza en varias iteraciones, cuyo número viene dado por el resultado de dividir el número de ficheros map por la propiedad io.sort.factor(por defecto 10). Por ejemplo, si se generaron 40 ficheros en la fase map, se realizarán cuatro iteraciones. En la primera se intercalarán 10 ficheros en uno sólo y así sucesivamente. Al finalizar se habrán obtenido cuatro ficheros que se pasan directamente a la fase reduce.

- Reduce: Se invoca a la función reduce por cada clave extraída del fichero generado en la fase Merge. La salida de esta fase puede ser grabada directamente en el HDFS, o bien a través de una función auxiliar para grabar la información con un formato determinado. Se genera un fichero en el HDFS por tarea reduce ejecutada.

3.2.2. HDFS.

HDFS es un sistema de ficheros pensado para almacenar grandes cantidades de información, del orden de terabytes o petabytes, tolerante a fallos y diseñado para ser instalado en máquinas de bajo coste. La información es dividida en bloques, que son almacenados y replicados en los discos locales de los nodos del clúster. Tiene muchas similitudes con otros sistemas de ficheros distribuidos, pero es diferente en algunos aspectos. Una diferencia notable es, que está enfocado a aplicaciones que siguen un modelo de una sola escritura y muchas lecturas, permitiendo relajar los requisitos de control de concurrencia, simplificando la coherencia de los datos, proporcionando como consecuencia un acceso de alto rendimiento. Otra cualidad única de HDFS es que parte de la suposición que, por lo general, es mejor ubicar la lógica de procesamiento cerca de los datos en lugar de mover los datos al espacio de aplicación.

HDFS se compone de un grupo de nodos interconectados, donde residen los archivos y directorios. Presenta una arquitectura master/worker o master/slave basada en un único nodo maestro, denominado NameNode, que maneja el espacio de nombres del sistema y regula el acceso de los clientes a los ficheros, redirigiéndolos a los nodos de datos que contienen la información, denominados DataNodes, que son los encargados de gestionar el almacenamiento en los discos locales del propio nodo. La Figura 46 muestra de forma gráfica la arquitectura HDFS.

Tanto el NameNode como los DataNodes son componentes software, diseñados para funcionar, de manera desacoplada, en máquinas genéricas a través de sistemas operativos heterogéneos. HDFS ha sido construido utilizando el lenguaje de programación Java, por lo tanto, cualquier máquina que soporte dicho lenguaje puede ejecutar HDFS. Una instalación típica consta de una máquina dedicada, donde se ejecutará el NameNode, y en cada una de las máquinas restantes que constituyen el clúster, se ejecutará un DataNode.

Una aplicación cliente, que desea leer un fichero en HDFS, debe contactar primero con el NameNode, para determinar en lugar en el cual está almacenada la información que requiere. En respuesta al cliente, el NameNode retorna el identificador del bloque más relevante y el nodo en el cual está almacenado. A continuación, el cliente contacta con el DataNode para recuperar la información requerida. Los bloques se encuentran almacenados en el sistema de ficheros local de la máquina, y el HDFS se encuentra en la parte superior de la pila del sistema operativo estándar (por ejemplo Linux). Una característica importante del diseño de este sistema de ficheros es, que la información nunca se mueve al NameNode. Toda la transferencia de información se produce directamente entre los clientes y los nodos de datos. La comunicación con el NameNode sólo implica transferencia de meta-información.

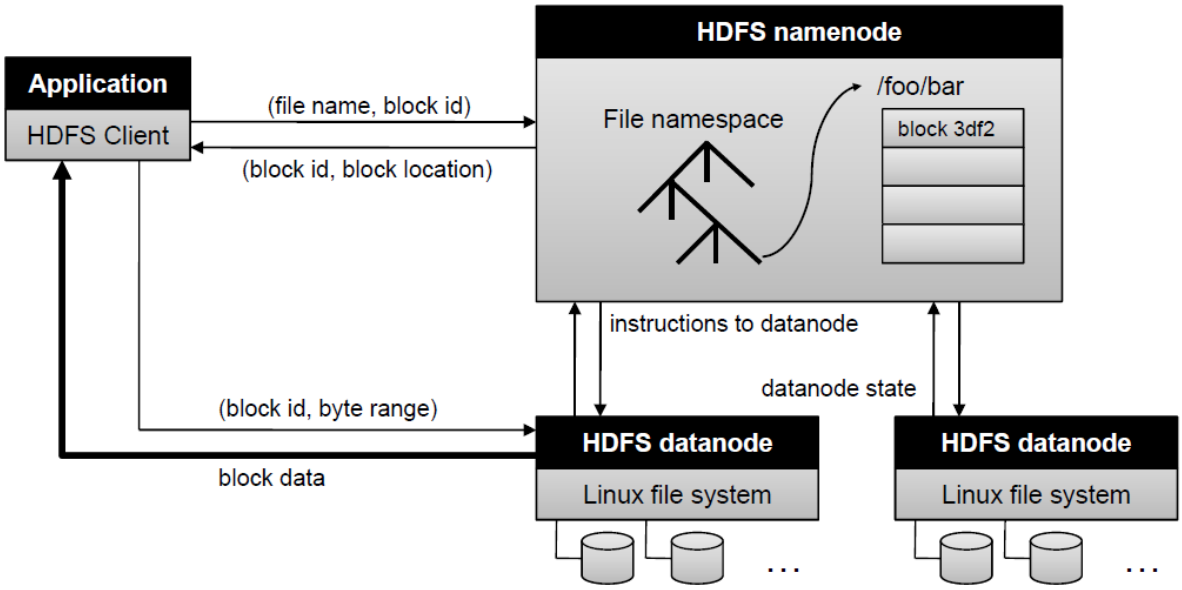


Figura 46: Arquitectura HDFS.

Los DataNodes facilitan periódicamente información del estado al NameNode, dado que este último no puede conectarse directamente con el primero. El Namenode simplemente se limita a responder a las peticiones realizadas por el DataNode, el cual mantiene un servidor de sockets abierto mediante el cual un cliente u otro DataNode puede leer o escribir información. El NameNode conoce el host o puerto para este servidor, proporcionando dicha información a los clientes interesados en contactar con un nodo de datos concreto.

HDFS presenta una estructura jerárquica de ficheros tradicional, en la cual un usuario puede crear directorios y almacenar ficheros bajo ellos. La jerarquía del espacio de nombres del sistema de ficheros es similar a otros sistemas existentes, permitiendo al usuario cambiar el nombre, la ubicación o eliminar ficheros.

Para que el sistema de ficheros sea tolerante a fallos, HDFS replica los bloques de ficheros. Una aplicación puede especificar el número de réplicas para un fichero en el momento que es creado, pudiendo ser cambiado en cualquier momento. El NameNode toma las decisiones relativas a la replicación de bloques. Por defecto, se mantiene tres replicas para cada fichero.

Uno de los objetivos principales de HDFS es dar soporte a ficheros de gran tamaño. El tamaño típico de un bloque de fichero en HDFS es 64Mb o 128Mb. Un fichero está compuesto de uno o varios bloques de 64⁄128Mb, y HDFS trata de colocar cada bloque en nodos de datos separados, distribuyendo la información a lo largo del clúster.

Los bloques no siempre pueden ser colocados de manera uniforme en los nodos de datos, lo que significa que el espacio disponible por uno o más nodos de datos puede estar infrautilizado. Otro caso común, que provoca que la distribución de los datos entre los diferentes nodos no esté balanceada, es la adición de nodos de datos al clúster. HDFS proporciona rebalanceo de bloques de datos utilizando diferentes modelos. Un modelo permite mover los bloques de un nodo de datos a otro, de forma automática, si el espacio libre en un nodo cae demasiado. Otro modelo permite crear, dinámicamente, réplicas adicionales para un determinado fichero, si se produce un aumento repentino de la demanda, rebalanceando otros bloques en el clúster. HDFS también proporciona comandos que permite realizar tareas de reajuste de forma manual.

Existe un elemento adicional denominado Secondary NameNode, cuyo objetivo es realizar periódicamente puntos de control (checkpoints) sobre los cambios que se van realizando en el sistema de ficheros.

3.2.3. Hadoop 2.0: YARN (Yet Another Resource Manager).

Uno de los problemas fundamentales que presenta Hadoop 1.0 es que solo admite un paradigma de programación: MapReduce. A pesar de que este modelo de programación es apropiado para el análisis de grandes conjuntos de datos, en ocasiones es necesario realizar otro tipo de análisis o sería más propicio utilizar otro tipo de software para analizar datos pero aprovechándonos de la ventaja que proporciona un clúster Hadoop. Para intentar solventar este inconveniente surge un nuevo componente fundamental dentro de Hadoop: YARN [Ref9, WWW]. Apache Hadoop YARN es un subproyecto de Hadoop en la Apache Software Foundation introducido en la versión Hadoop 2.0 que separa la gestión de recursos de los componentes de procesamiento. YARN surge para corregir el inconveniente principal de Hadoop y permitir una gama más amplia de modelos de programación para analizar

los datos almacenados en el HDFS más allá de MapReduce. La arquitectura de Hadoop 2.0 basada en YARN provee una plataforma de procesamiento más general y no restringida a MapReduce.

En Hadoop 2.0, YARN toma las capacidades de gestión de los recursos que residían en MapReduce y las empaqueta para que puedan ser utilizados por los nuevos motores de procesado. Esto también simplifica la tarea de MapReduce en únicamente hacer lo que mejor sabe hacer, tratar datos. Con YARN, se permite ejecutar varias aplicaciones en Hadoop, todos compartiendo una gestión común de los recursos. MapReduce se convierte ahora en una librería Hadoop es decir una aplicación que reside en Hadoop y deja la gestión de recursos del clúster para el componente YARN. En la Figura 47 se muestra la evolución de los principales componentes de Hadoop.

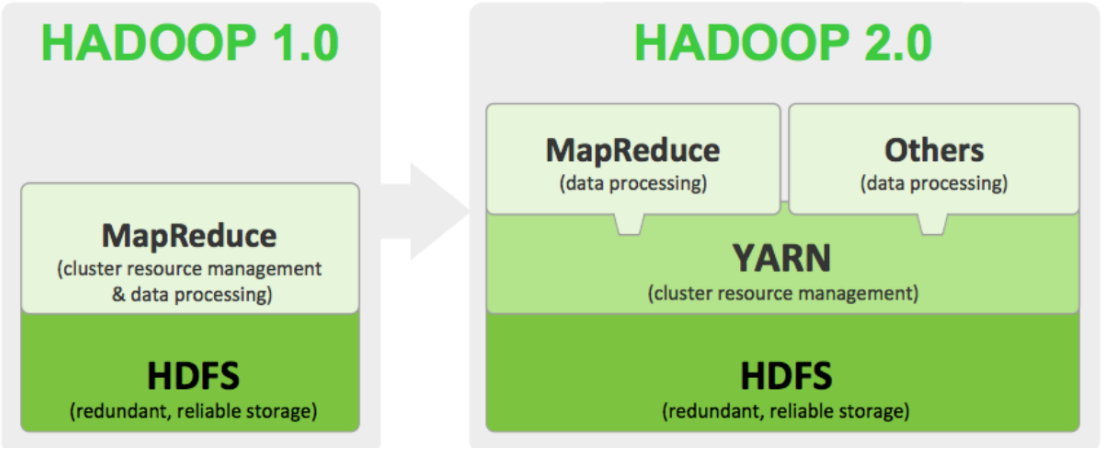


Figura 47: Evolución de Hadoop 1.0 a 2.0.

La aparición de YARN provoca el desarrollo de nuevas herramientas que cubren múltiples necesidades que únicamente con MapReduce no se podían completar. En la Figura 48 aparecen algunas de estas tecnologías.

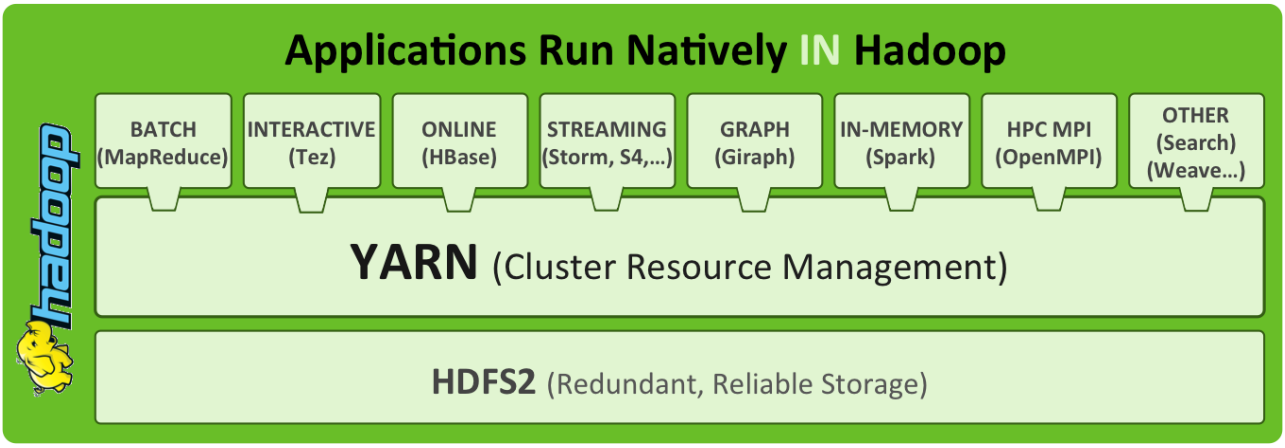


Figura 48: Ejemplo de nuevas aplicaciones YARN.

Hadoop 2.0: Arquitectura.

La idea fundamental de YARN es la de separar las dos mayores responsabilidades del JobTracker: la gestión de los recursos y la planificación/monitorización de las tareas en dos servicios separados, para ello tendremos dos nuevos componentes en YARN: un ResourceManager global y un ApplicationMaster por aplicación (AM).

Surgen así el ResourceManager para el master (sustituyendo al JobTracker) y un NodeManager (sustituyendo al TaskTracker) por cada slave. Forman el nuevo, y genérico, sistema de gestión de aplicaciones de una manera distribuida. Además surge un componente llamado container que representa los recursos disponibles en cada nodo del clúster.

El ResourceManager es la última autoridad que arbitra los recursos entre todas las aplicaciones en el sistema. El ApplicationMaster por aplicación es una entidad específica que se encarga de negociar los recursos con el ResourceManager y trabajar con los NodeManager para ejecutar y supervisar las tareas que lo componen.

El ResourceManager está conectado a un planificador (Scheduler) que es responsable de la asignación de recursos a las diversas aplicaciones que se ejecutan con limitaciones conocidas de capacidades, las colas, etc. El Scheduler es un planificador puro en el sentido de que no realiza ningún monitoreo o seguimiento del estado de la aplicación. El Scheduler realiza su función de planificación en base a las necesidades de recursos de las aplicaciones fundamentándose en la noción abstracta de un container de recursos que incorpora elementos de recursos como la memoria, CPU, disco, red, etc.

El NodeManager se encuentra en cada uno de los equipos esclavos y es responsable del lanzamiento de los container de las aplicaciones, el seguimiento del uso de recursos (CPU, memoria, disco de red) y de informar del mismo al ResourceManager.

El ApplicationMaster tiene la responsabilidad de negociar los containers de recursos necesarios desde el Scheduler, de realizar un seguimiento de su estado y de los

progresos de ejecución. Desde la perspectiva del sistema, el propio ApplicationMaster se ejecuta como un container normal. En la Figura 49 se presenta la arquitectura comentada.

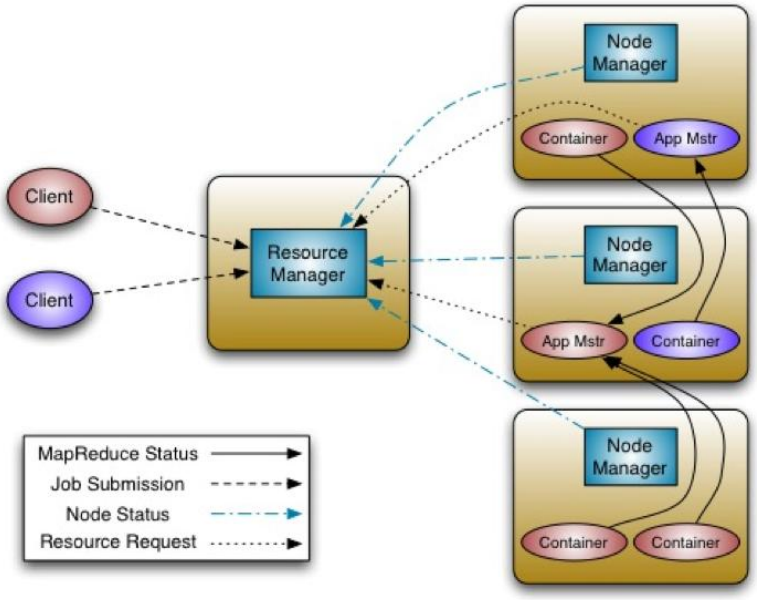


Figura 49: Arquitectura Hadoop 2.0.

A pesar de que los componentes evolucionen, una de las virtudes de YARN, es que mantiene el esquema de gestión de recursos utilizado con MapReduce lo que genera que Hadoop 2.0 esté totalmente integrado en este paradigma de programación.

Hadoop 2.0: Ciclo de aplicación.

La ejecución de una aplicación en Hadoop 2.0 consta de los siguientes pasos:

- Presentación de la solicitud.
- Generación del ApplicationMaster para la aplicación.
- Ejecución de la aplicación gestionada por el ApplicationMaster.

A continuación vamos a secuenciar los pasos que sigue la ejecución de una aplicación (todos los pasos están ilustrados en la Figura 50)[Ref10, WWW]:

1. Un programa cliente envía la solicitud, incluyendo las especificaciones necesarias para poner en marcha el propio ApplicationMaster de la aplicación.
2. El ResourceManager asume la responsabilidad de negociar un container en el que debe comenzar el ApplicationMaster, seguidamente ejecuta el ApplicationMaster.

3. El ApplicationMaster, una vez arrancado, se registra con el ResourceManager . El registro permite que el programa consulte detalles de los recursos al ResourceManager.
4. Durante un funcionamiento normal de la ejecución, el ApplicationMaster negocia containers de recursos adecuados a través del protocolo de recursos-petición.
5. Cuando la asignación de contenedores es satisfactoria, la ApplicationMaster lanza el container, proporcionando las especificaciones de la ejecución del container al NodeManager. La información de ejecución, por lo general, incluye la información necesaria para permitir que el container se comunique con el mismo ApplicationMaster.
6. El código de la aplicación que se ejecuta dentro del container proporciona la información necesaria (progreso, estado, etc.) a su ApplicationMaster a través de un protocolo específico de la aplicación.
7. Durante la ejecución de la aplicación, el cliente que presentó el programa se comunica directamente con el ApplicationMaster para consultar el estado, actualizaciones de progreso, etc. A través de un protocolo específico de la aplicación.
8. Una vez que la solicitud es completada, y todo el trabajo necesario se ha finalizado, el ApplicationMaster anula el registro con el ResourceManager y se apaga, lo que permite al contenedor ser reutilizado por otra solicitud.

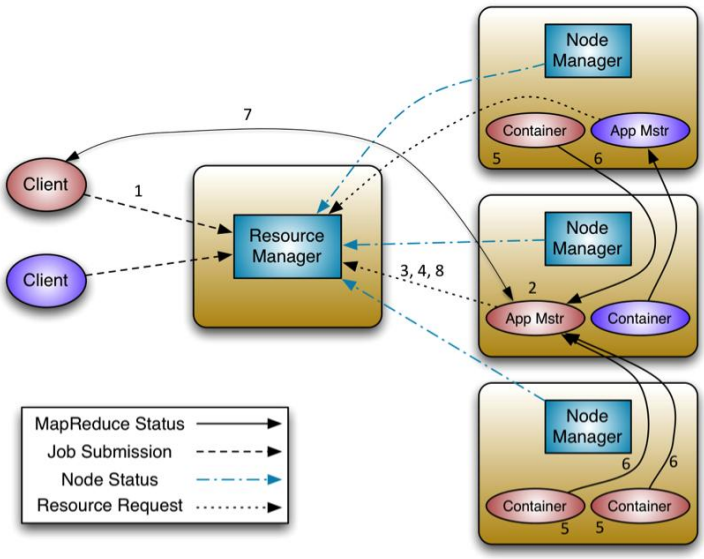


Figura 50: Ejecución de una aplicación Hadoop 2.0.

3.2.4. Ecosistema Hadoop.

El proyecto Hadoop consta de una serie de subproyectos, que vienen a complementar su funcionalidad profundizando en aspectos como el tratamiento, flujo e importación de datos, la monitorización de trabajos, etc. Existen multitud de proyectos relacionados con Hadoop que completan distintas necesidades (ver Figura 51). La mayoría son dirigidos por Apache aunque empresas privadas como Cloudera o Hortonworks trabajan desarrollando todo este tipo de plataformas.

A continuación, vamos a presentar los proyectos relacionados con Hadoop más importantes:

- Ambari: Una herramienta basada en web para el aprovisionamiento, administración y seguimiento de clústeres Apache Hadoop, que incluye soporte para Hadoop HDFS, Hadoop MapReduce, Colmena, HCatalog, HBase, ZooKeeper, Oozie, Pig y Sqoop. Ambari también proporciona un panel de control para la visualización del estado del clúster, así como la capacidad de ver aplicaciones como MapReduce, Pig y Colmena con el objetivo de evaluar su rendimiento de una manera sencilla.

- Avro: se trata de un sistema de serialización de datos que provee numerosas estructuras de datos, un formato de datos binario compacto y rápido, un archivo contenedor para almacenar datos persistentes y una sencilla integración con lenguajes dinámicos.

- Cassandra: Apache Cassandra es una base de datos distribuida de segunda generación altamente escalable, que reúnele diseño totalmente distribuido de Dynamo y el modelo de datos basado en ColumnFamily de Bigtable. Cassandra se usa en Facebook, Digg, Twitter, Mahalo, Ooyala, SimpleGeo, Rackspace, y otras empresas que necesitan de una base de datos con alta escalabilidad, disponibilidad y tolerancia a fallos.

- Chukwa: es un sistema de recopilación de datos de código abierto para el seguimiento de grandes sistemas distribuidos. Chukwa hereda la escalabilidad y robustez de Hadoop. Además incluye un conjunto de herramientas flexibles y potentes para la visualización, seguimiento y análisis de resultados para hacer el mejor uso de los datos recogidos.

- HBase: Una base de datos escalable, distribuida que soporta el almacenamiento de datos estructurados en tablas. Permite la realización de tablas a partir de ficheros de datos.

- Hive: facilita la consulta y gestión de grandes conjuntos de datos que residen en almacenamiento distribuidos. Hive proporciona un mecanismo para la ver la estructura de los datos utilizando un lenguaje similar a SQL llamado HiveQL.

- Mahout: se trata de un software libre centrado en la implementación de algoritmos de machine learning distribuidos.

- Pig: Apache Pig es una plataforma para el análisis de grandes conjuntos de datos que se caracteriza por un lenguaje de alto nivel para la creación de los programas de análisis de datos, junto con la infraestructura necesaria para la evaluación de estos programas. La propiedad más importante de los programas Pig es que su estructura es susceptible de una paralelización sustancial, lo que a su vez permite manejar grandes conjuntos de datos. El siguiente apartado del trabajo irá dedicado esta herramienta con el objetivo de mostrar todas sus posibilidades en el análisis de ficheros de datos.

- Spark: proporciona un motor de cálculo rápido y general para datos Hadoop. Spark proporciona un modelo de programación sencillo y expresivo que soporta una amplia gama de aplicaciones, incluyendo ETL, machine learning, procesamiento de flujo, y computación gráfica.

- ZooKeeper: es un servicio centralizado construido para mantener la información de configuración, proporcionar sincronización distribuida y la prestación de servicios de grupo. Todos estos tipos de servicios se utilizan de una forma u otra por las aplicaciones distribuidas. Esta plataforma es utilizada en Storm como ya se describió durante el Capitulo 2.

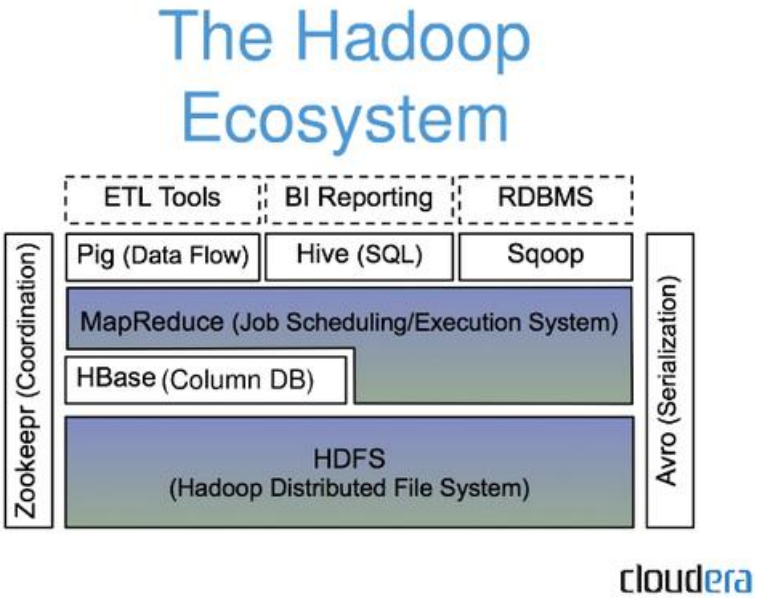


Figura 51: Ecosistema Hadoop.

3.3. Instalación del servicio virtualizado de Hortonworks.

En el presente capítulo se explica la instalación de Hadoop en un entorno Linux instalado en una maquina virtual. Como se podrá comprobar, el proceso de instalación y configuración es relativamente extenso y complejo (Ver *Anexo A*). Para evitar el proceso de instalación al usuario de Hadoop, empresas privadas dedicadas al desarrollo de la plataforma, han creado versiones pre-configuradas de Hadoop donde únicamente se importa un servicio virtualizado a través de una maquina virtual (por ejemplo VirtualBox) y se tendría Hadoop ya instalado en muy pocos minutos. Además de la última versión de Hadoop instalada, este servicio tiene ya instaladas numerosas herramientas relacionadas con Hadoop. Las dos principales empresas proveedoras de este tipo de tecnología son Hortonworks y Cloudera.

En este trabajo, se va a instalar la versión pre-configurada de Hadoop proporcionada por Hortonworks. Se denomina Hortonworks Sandbox y puede ser descargada desde la página oficial de Hortonworks [Ref11, WWW]. Este servicio proporciona un entorno Hadoop con trece componentes adicionales. Las especificaciones técnicas de Hortonworks Sandbox 2.1 se pueden observar en la Figura 52:

Apache Hadoop	2.4.0
Apache Hive	0.13.0
Apache HBase	0.96.1
Apache Pig	0.12.1
Apache Storm	0.9.1
Apache Solr	4.8
Apache Falcon	0.5
Apache Sqoop	1.4.5
Apache Flume	1.4.0
Apache Oozie	4.0.0
Apache Ambari	1.5.1
Apache Mahout	0.9.0
Apache ZooKeeper	3.4.5
Apache Knox	0.4

Figura 52: Especificaciones técnicas Hortonworks Sandbox 2.1.

Esta herramienta se ejecuta en un entorno Linux dentro de la máquina virtual de nuestro computador y es accesible a través de nuestro navegador. En la Figura 53 se puede visualizar el esquema de funcionamiento.

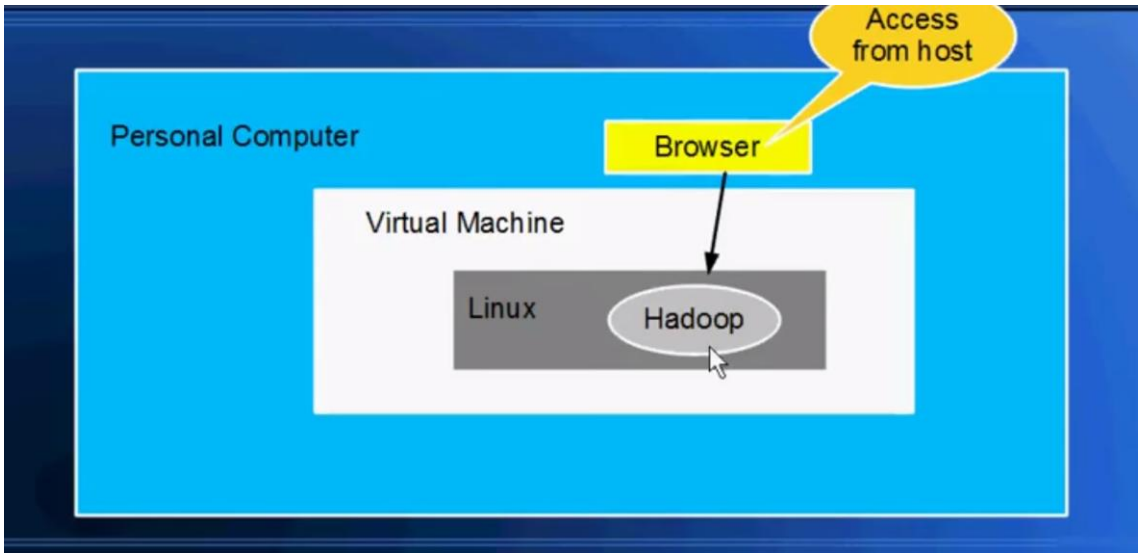


Figura 53: Esquema de funcionamiento de la Sandbox.

La principal ventaja de este tipo de servicios es que ofrecen un entorno Hadoop de una manera muy sencilla y sin instalación. En muy pocos minutos se puede tener Hadoop operativo y todas las herramientas adheridas. A esta virtud, se le suma una interfaz gráfica para trabajar con estas herramientas, algo que no ofrece Hadoop por sí mismo. En contra, esta plataforma solo admite un modo de ejecución local o Single-node, por tanto, no permite aprovechar el modo de ejecución distribuido de Hadoop. Este software está meditado para ofrecer al usuario Hadoop una plataforma de pruebas donde poder desarrollar sus programas de análisis de datos para, posteriormente, ser utilizados en un clúster Hadoop.

3.3.1 Instalación.

Para instalar el servicio virtualizado es necesario tener instalado un software de virtualización. En el presente proyecto se ha optado por el software Virtualbox (ver Figura 54) pero Hortonworks también ofrece la posibilidad de un servicio virtualizado para el software de virtualización VMWare.

Se puede descargar virtualBox desde su página oficial [Ref12, WWW]. Ofrece versiones para 4 sistemas operativos distintos: Windows, Linux, OS X y Solaris. Una vez descargado, se instala siguiendo el proceso habitual.

Figura 54: Icono VirtualBox.

Cuando se abre VirtualBox se observa una ventana de este tipo:

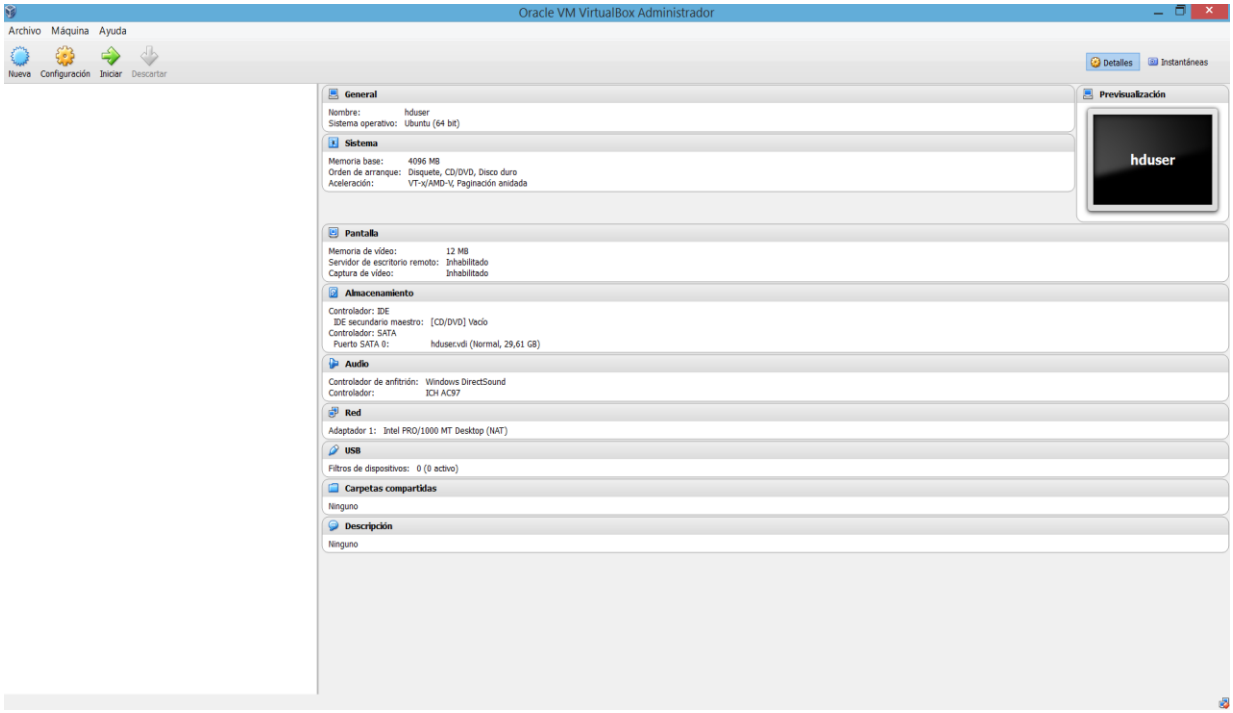


Figura 55: Ventana principal VirtualBox.

Una vez instalado VirtualBox, se puede descargar el servicio virtualizado para VirtualBox [Ref13, WWW].

Con el servicio virtualizado descargado, se hace click sobre Archivo>Importar servicio virtualizado:

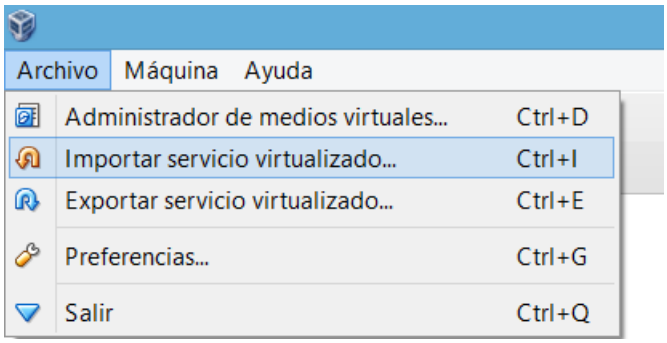


Figura 56: Ventana VirtualBox.

Se abrirá la ventana de la Figura 57:

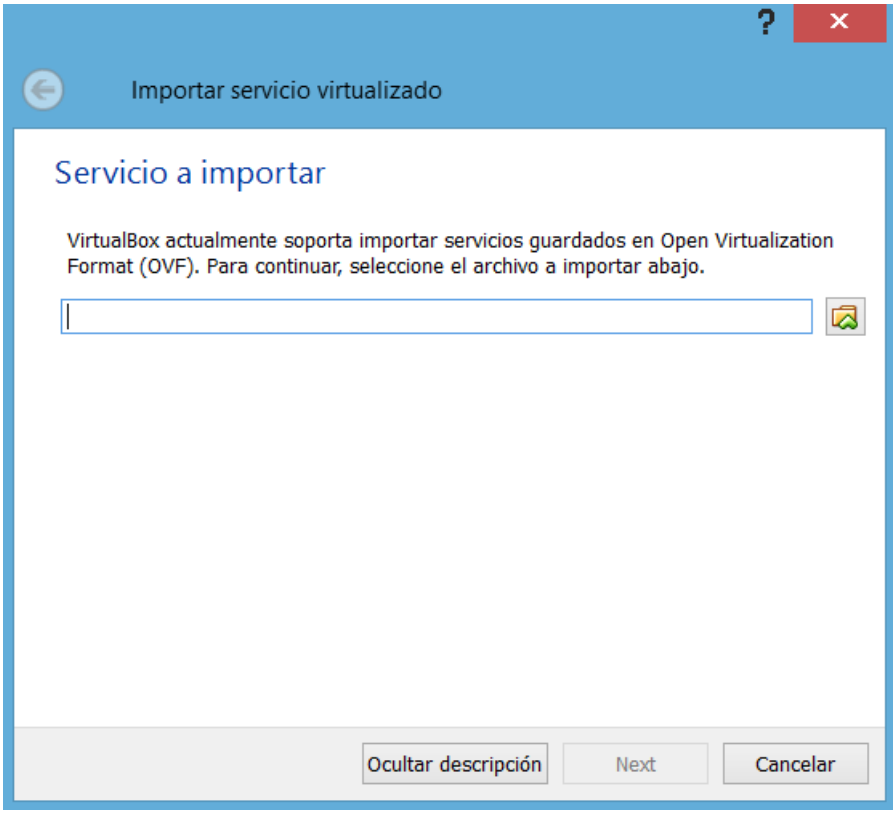


Figura 57: Ventana VirtualBox.

Haciendo click sobre el botón  se podrá seleccionar el servicio virtualizado previamente descargado. Se pulsa Next y se abrirá la ventana de la Figura 58.

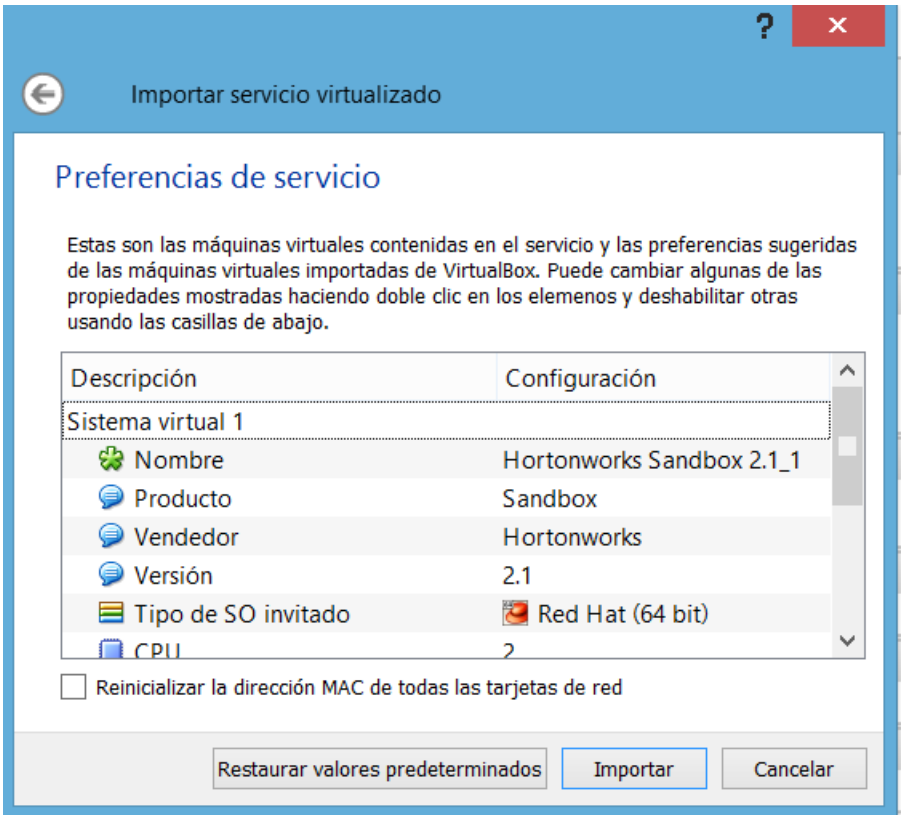


Figura 58: Ventana VirtualBox.

Pulsando Importar y se crearía la máquina virtual (ver Figura 59).

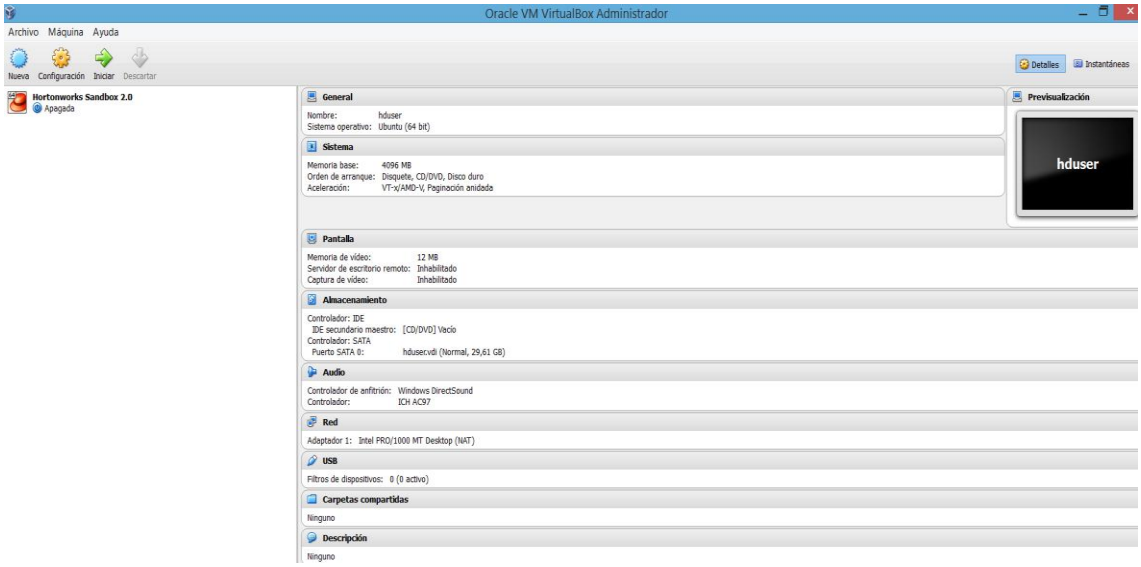


Figura 59: Ventana VirtualBox.

Para iniciar la máquina virtual se pulsa Iniciar y una vez cargada se mostrará la pantalla de la figura 60.

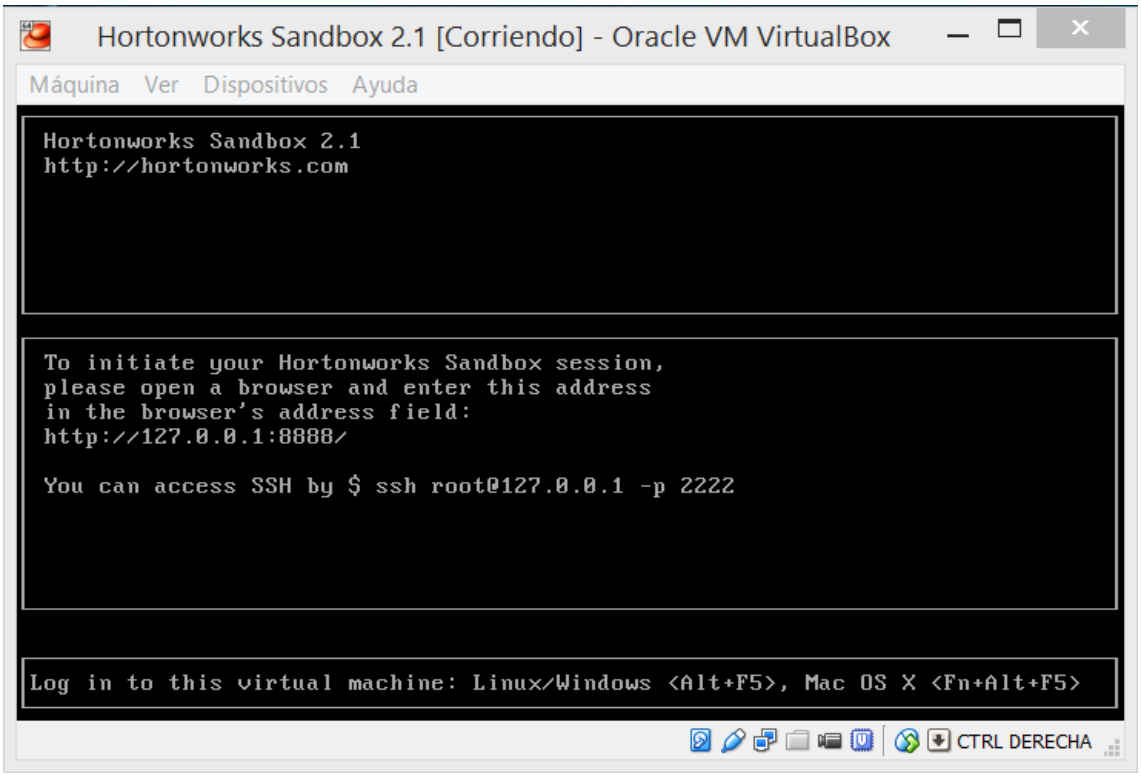


Figura 60: Ventana servicio virtualizado.

Para empezar a utilizar la plataforma, basta con escribir la url que se muestra. Se puede ver la ventana principal de la Hortonworks Sandbox en la figura 61:

Figura 61: Ventana principal Hortonworks Sandbox 2.1.

3.3.2 Características del servicio virtualizado.

Como se puede comprobar en la figura 61, el servicio virtualizado muestra una interfaz de usuario donde las herramientas para trabajar se encuentran en la barra superior.

Figura 62: Herramientas de la Hortonworks Sandbox.

Se van a presentar las 4 primeras herramientas ya que son las más utilizadas a la hora de realizar el código para analizar datos, como se señala en la Figura 62.

Se comienza con el File Browser, que se trata de un navegador de archivos que permite subir/descargar archivos al HDFS. Una vez se ha subido un archivo al HDFS se nos añadirá a la lista de archivos. En la Figura 63, se observa el aspecto del File Browser.

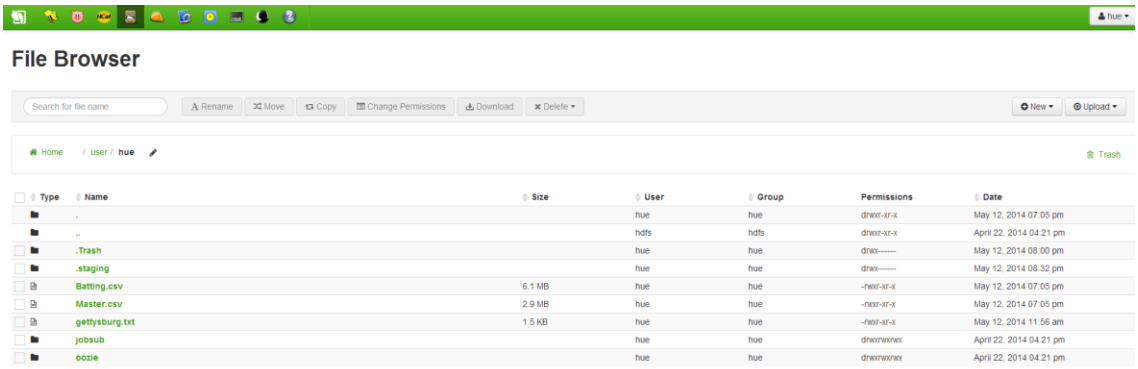


Figura 63: File Browser.

La segunda herramienta se denomina HCatalog. Se trata de un gestor de tablas y almacenamiento. Su función principal es la creación de tablas a partir de los archivos almacenados en el HDFS para facilitar su manejo con herramientas como Pig o Hive. En la Figura 64 se muestra el aspecto visual de HCatalog.

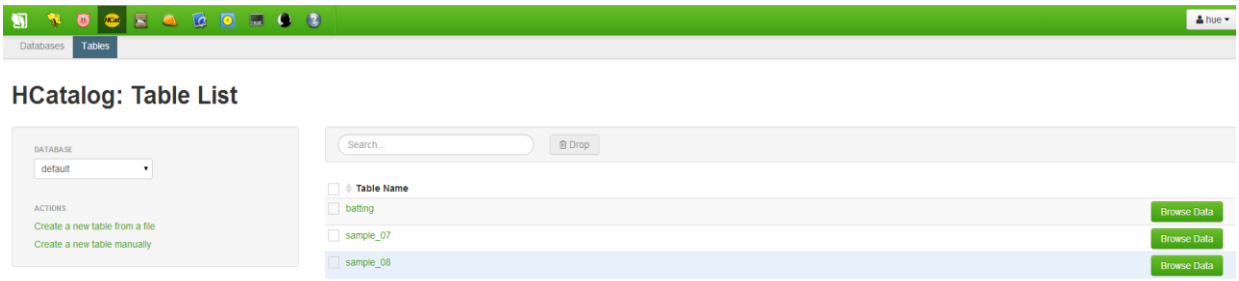


Figura 64: HCatalog.

En tercer lugar se encuentra la plataforma Pig. Un lenguaje de alto nivel creado para procesar ficheros de una manera sencilla y ágil. La Hortonworks Sandbox permite crear Scripts Pig y mantenerlos almacenados. También facilita una ayuda en la que se muestra la sintaxis de algunas de las funciones Pig más utilizadas así como un editor de texto muy útil para el desarrollo de programas Pig. Otra característica importante es que este servicio permite la creación de UDF (User defined functions) que se trata de funciones creadas por el usuario en un lenguaje distinto a Pig para realizar algún tipo de procesado que Pig no puede realizar. Las UDF se pueden realizar en 4 lenguajes de programación distintos: Python, Jython, Java y Ruby. En la Figura 65 se ve el aspecto visual de Pig.

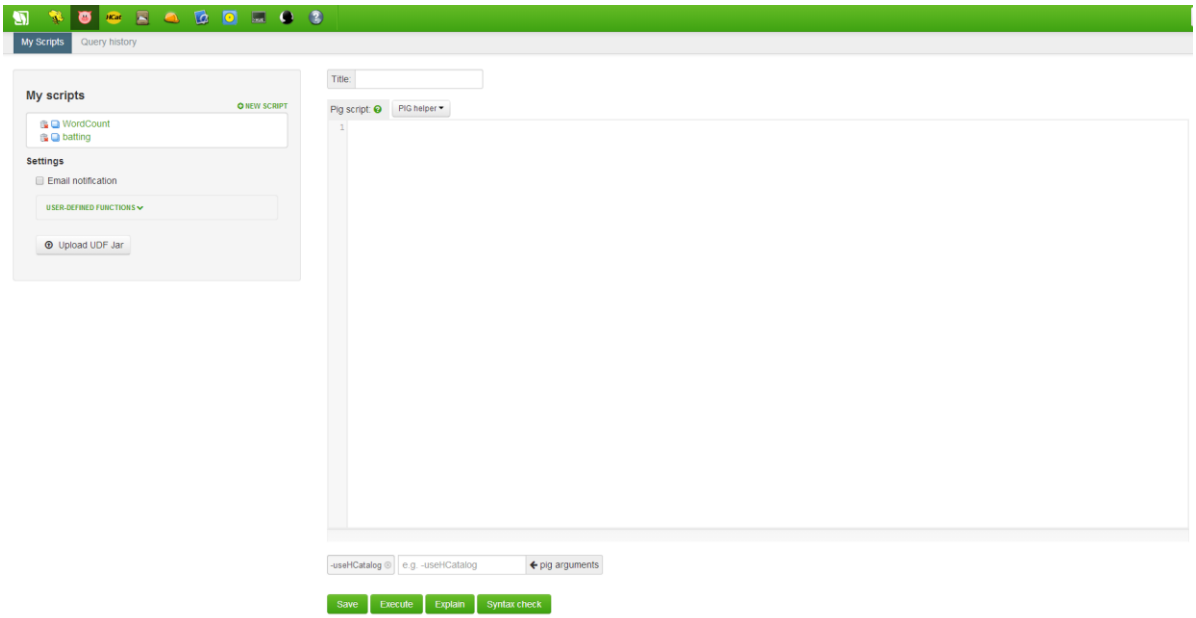


Figura 65: Apache Pig.

Por último, se tiene Apache Hive. Esta herramienta permite desarrollar programas de

consulta de características de tablas o de archivos almacenados en la base de datos con un lenguaje muy parecido al SQL de bases de datos tradicionales. En la Figura 66 se presenta la tecnología.

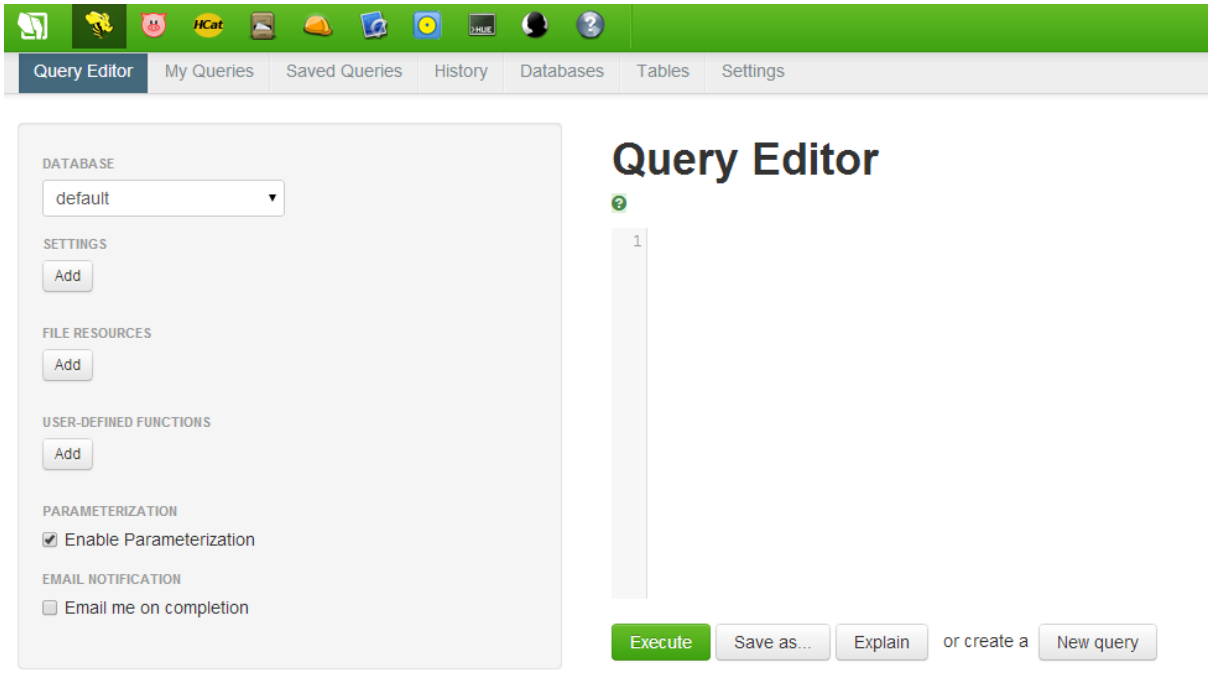


Figura 66: Apache Hive.

3.4. Hadoop Java API.

Antes de proceder con el análisis de datos mediante JAVA MapReduce es necesario conocer los aspectos principales de la API para Hadoop. En este apartado se va a presentar la estructura básica de un programa JAVA para Hadoop MapReduce. Como se ha explicado ampliamente en capítulos anteriores, el paradigma MapReduce esta constituido por dos tareas fundamentales (Map y Reduce), por tanto, cada una de estas tareas debe ser implementada cuando se realiza el código JAVA. Además de estas dos tareas, existen otras etapas intermedias que no son imprescindibles. En la siguiente figura 67 se muestra esquemáticamente el ciclo que sigue un programa basado en Hadoop MapReduce:

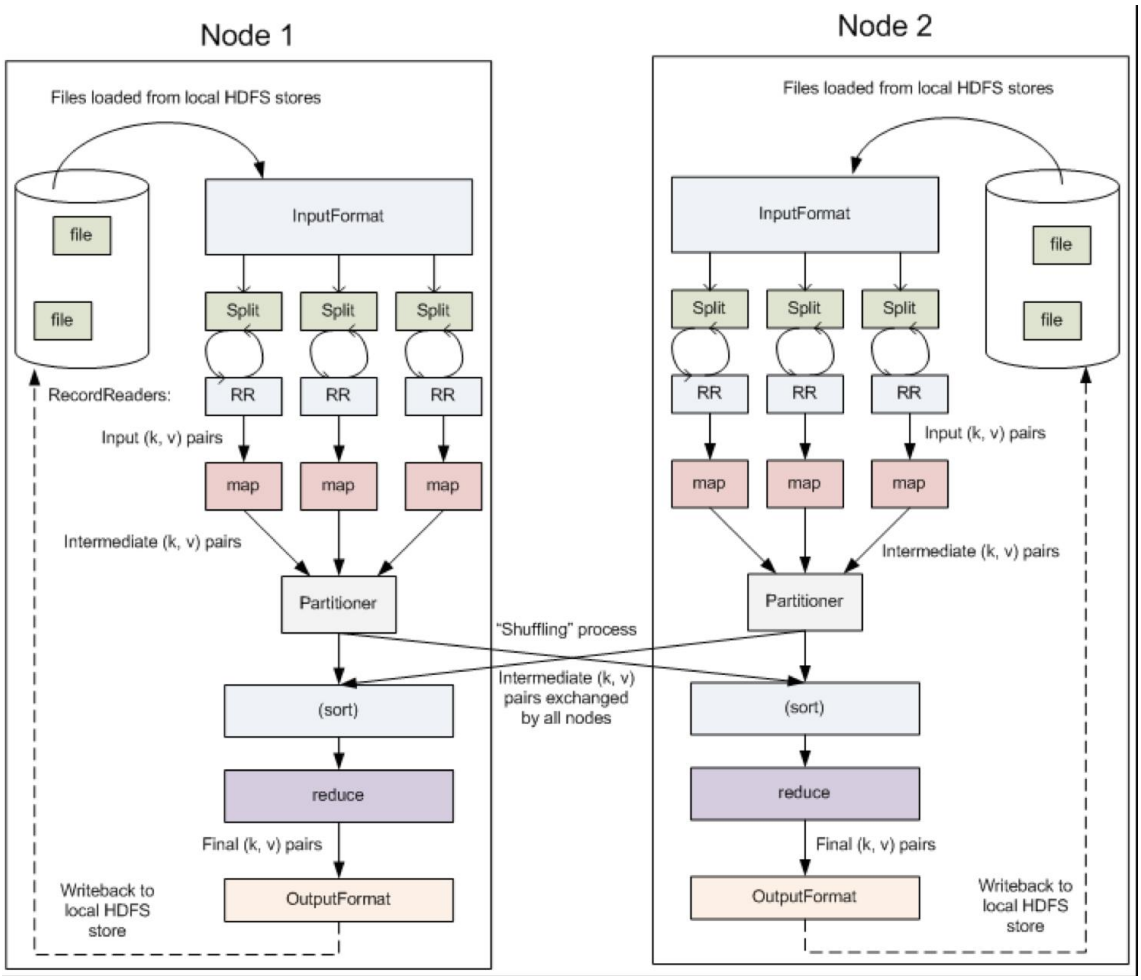


Figura 67: Ciclo de programa MapReduce.

Los programas realizados en JAVA MapReduce para Hadoop están compuestos por tres clases principalmente:

- Clase Map: Hereda de la clase Mapper y comprende métodos encargados de realizar la tarea Map. En esta acción, se transforman las entradas en una serie de parejas clave/valor intermedias.

- Clase Reduce: Hereda de la clase Reducer e incluye métodos encargados de realizar la tarea Reduce. Toma como entrada las parejas clave/valor provenientes de la fase Map.

- Clase principal: Contiene el método main que engloba parámetros de configuración, como tipos de salida, ruta de la entrada etc. En esta clase se compone un Job y se ejecuta. Esta clase además engloba a las dos clases anteriores estáticas, una estructura típica cuando se implementa una API en JAVA.

Clase Map

Como se ha explicado en capítulos anteriores, en esta etapa se generan una serie de valores intermedios que serán la entrada para la etapa Reduce. La entrada recibida en esta etapa depende en gran parte de la implementación de InputFormat utilizada. Por defecto, la clave de entrada simboliza la línea del fichero y el valor el número de la palabra (elemento que encuentra separado por espacios en blanco). Si bien se pueden realizar otras implementaciones si es necesario.

Para cada entrada, se ejecuta el método `map(LongWritable key, Text value, Mapper<LongWritable, Text.Text, IntWritable>.Context context)` y se genera una pareja clave/valor de salida. A continuación se presenta el esqueleto de esta clase:

```
public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable>
        {


          public void map(LongWritable key, Text value,
Mapper<LongWritable, Text, Text, IntWritable>.Context context) throws
IOException, InterruptedException {

              context.write(clave, valor);
            }
          }
        }
```

Como se puede percibir, se compone principalmente de un método map donde podemos realizar cualquier operación utilizando JAVA a cada entrada. Cabe mencionar que mediante el objeto Context podemos comunicarnos con el sistema MapReduce, por ejemplo, escribir como salida de esta etapa las parejas clave/valor. De todos los objetos mencionados existen numerosos métodos que otorgan una mayor funcionalidad a nuestro programa MapReduce.

Clase Reduce

En esta última etapa se reducen los resultados provenientes de la clase map en una lista de parejas clave/valor más pequeñas. El formato de salida depende de OutputFormat. A continuación, se presenta la estructura de esta clase:

```
    public static class Reduce extends Reducer<Text, IntWritable,
Text, IntWritable> {
          public void reduce(Text key, Iterable<IntWritable> values,
Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws
IOException, InterruptedException {

            context.write(clave, valor);
}
```

```
        }
    }
```

Como se puede comprobar en el código, se vuelve a utilizar el objeto context para escribir los datos de salida.

Clase principal o de configuración

En esta clase se establece toda la configuración necesaria para la correcta ejecución del programa. Además, se crea un objeto job que representa la ejecución del código. La configuración esencial se muestra seguidamente:

```
public class myPrograma {
      public static void main(String[] args) {

                Configuration conf = new Configuration();

                Job job = new Job(conf, "myPrograma");
                job.setJarByClass(myPrograma.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(myPrograma.Map.class);
                job.setReducerClass(myPrograma.Reduce.class);


                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.waitForCompletion(true);
            }
```

Como se puede observar se establece toda la configuración imprescindible para la correcta ejecución del programa. Es necesario crear un objeto Job e indicar el formato de salida de las parejas clave/valor. Asimismo, es necesario indicar los nombres de las clases Map y Reduce. Por ultimo, se indica la procedencia de los datos de entrada y donde se escriben los datos de salida mediante addInputPath y setOutputPath.

El objeto Job permite invocar numerosos métodosrelacionados con la configuración, los principales son los siguientes:

```
public void setInputFormatClass(..);
public void setOutputFormatClass(..);
public void setMapperClass(..);
public void setCombinerClass(..);
public void setReducerClass(...);
public void setPartitionerClass(..);
public void setMapOutputKeyClass(..);
public void setMapOutputValueClass(..);
public void setOutputKeyClass(..);
public void setOutputValueClass(..);
public void setJobName(String name);
public float mapProgress();
public float reduceProgress();
```

```
public boolean isSuccessful();
public void killJob();
public void submit();
public boolean waitForCompletion(..);
```

Se puede consultar la API completa en la página oficial de Hadoop [Ref14, WWW].

3.5. Apache Pig.

Pig es una plataforma de alto nivel para procesar conjuntos de datos en paralelo mediante Hadoop, haciendo uso de su HDFS y de Hadoop MapReduce. Provee de un lenguaje de programación denominado Pig Latin que permite crear programas MapReduce en un lenguaje similar a SQL. Pig supone un nivel de abstracción bastante más alto que JAVA MapReduce. No obstante y como se ha comentado en anteriores capítulos, Pig permite la creación de UDF (User Defined Functions) programados en JAVA, JavaScript, Python, Ruby y Groovy.

Pig Latin es un lenguaje de flujo de datos o “dataflow”. Esto implica que el usuario es capaz de describir como datos de una o múltiples entradas son leídos, procesados y almacenados en paralelo. Estos flujos de datos pueden ser sencillos, por ejemplo, un programa que únicamente lee y almacena datos. Aunque pueden ser complejos flujos de trabajo con múltiples puntos donde las entradas son cruzadas o donde la entrada es troceada para ser analizada por diferentes operadores. Matemáticamente, un script de Pig latín describe un grafico acíclico dirigido o DAG (Directed acyclic graph) [Ref15,WWW]. Un DAG es un grafo dirigido que no tiene ciclos lo que significa que para cada vértice v, no hay un camino directo que empiece y termine en v. En la siguiente figura se muestra un ejemplo de DAG:

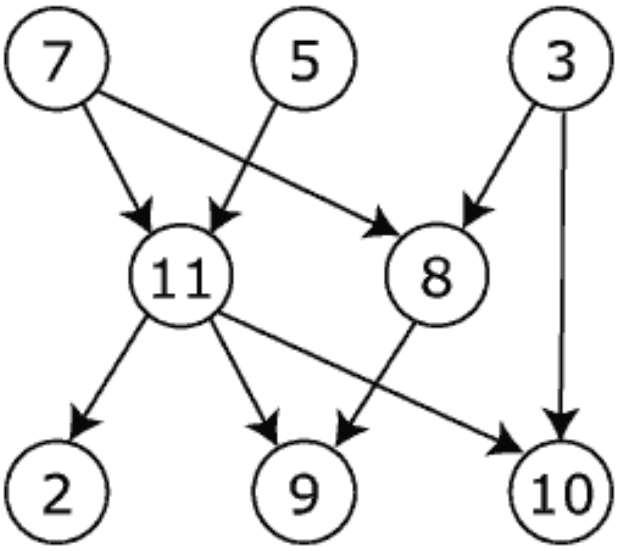


Figura 68: Ejemplo de DAG.

Esto significa que Pig Latin es muy diferente de los lenguajes de programación orientados a objetos que se basan en el control de flujo. En Pig no existen las sentencias if (aunque es posible realizar condiciones lógicas) o los bucles for ya que está enfocado al flujo de datos y no de control [Ref16].

Como se ha comentado al inicio de este aparato Pig luce similar a SQL y después de un estudio superficial pueden parecer más similares aún, sin embargo, en su filosofía son muy distintos. SQL es un lenguaje de búsqueda y tiene como fin permitir al usuario describir cuestiones que quiere resolver. Pig, sin embargo, posibilita que el usuario describa exactamente como procesar unos datos de entrada. Además, SQL está orientado a contestar una consulta. Si el usuario necesita realizar más consultas debe escribir por separado cada una de ellas guardando los datos intermedios en tablas temporales. Pig sin embargo, está diseñado para tratar con grandes series de datos por tanto no necesitamos guardar datos temporales.

Esta plataforma fue originalmente desarrollada en la empresa Yahoo durante el 2006. En el 2007, fue donado a la asociación Apache Software Foundation. Esto posibilita a que cualquier usuario pueda descargar la herramienta.

Tipos de Datos

Los tipos de datos en Pig pueden dividirse en dos categorías: escalares o simples, que contienen un único valor, y tipos complejos, que contienen otros tipos de datos.

Escalares o simples

- Int: Se trata de valores enteros y son representados en las interfaces por java.lang.Integer. Guarda un valor con signo de 32 bits.

- Long: Enteros de 64 bits. Están representados por las interfaces java.lang.Long.

- Float: Numeros en punto flotante, es decir, con parte entera y decimal. Guarda números decimales de 32 bits y esta representado por la interfaces java.lang.Float.

- Double: Numeros decimales de 64 bits. Están representados por las interfaces java.lang.Double.

- Chararray: Se trata de un "String"o cadena de caracteres en formato Unicode UTF-8. Estan representados por las interfaces java.lang.String.

- Datetime: Se trata de fechas. Pig utiliza las clases org.joda.time.DateTime.

- Boolean: Representa los valores booleanos True/False.

Complejos

Pig tiene tres tipos de datos complejos: maps (mapa), tuples (tuplas), y bags (bolsas). Todos estos tipos pueden contener datos de cualquier otro tipo. Es decir, es posible tener un map donde un campo es una bag que contiene una tuple donde uno de los campos es un map.

- Map: Se trata de parejas clave/valor que pueden contener cualquier tipo de dato. Se forma mediante corchetes para delimitar los campos y una almohadilla para la separación entre la clave y el valor. Por ejemplo, ['nombre'# 'antonio','edad'#'34'] creara un map de dos claves 'nombre' y 'edad'. El primer dato es un chararray y el segundo un int.

- Tuple (Tupla): Una tupla en Pig es una colección de datos ordenados y de longitud fija. Estan divididas en campos (fields) donde cada campo contiene un elemento que puede ser de cualquier tipo. La tuplas usan paréntesis y comas para delimitar los campos. Por ejemplo ('Antonio', 30) describe una tupla con dos campos.

- Bag (Bolsa): Es definida como una colección desordenada de tuplas. Como están desordenadas, no es posible referenciar tuplas en una bolsa por su posición. Por ejemplo, {('Antonio', 30), ('Francisco', 20)} construirá una bolsa con dos tuplas, cada una con dos campos.

Operadores Pig.

Este subapartado está dedicado a presentar los principales operadores Pig. Solo se presentan los más utilizados ya que el gran número de operadores y funciones que existen en Pig provoca que no sea posible presentarlos todos en este documento. Existen múltiples clases de operadores y funciones, las más utilizadas son: Funciones de entrada/salida, Operadores de diagnóstico, operadores relacionales, funciones matemáticas, funciones de evaluación, funciones de Strings y funciones para DateTime.

- Funciones de entrada/salida de datos: Permiten cargar y almacenar datos según su delimitador de campos.

Operador	Descripción	Sintaxis
Load	Permite cargar uno o múltiples ficheros de datos.	LOAD 'data' [USING function] [AS schema]
Store	Guarda una variable (datos) en una ruta específica.	STORE alias INTO 'directory' [USING function]

Se puede establecer cualquier delimitador pero por defecto se establece el tabulador (/t). El campo SCHEMA representa el esquema que siguen los datos separados por el delimitador. Por defecto, se le asigna el valor $X siendo X el número del campo, es decir, el primer campo si no establecemos un schema sería el $0. Para estos operadores Pig no es Case Sensitive.

- Operadores de diagnóstico: Tienen como objeto ayudar al usuario a desarrollar scripts Pig.

Operador	Descripcion	Sintaxis
Dump	Muestra por pantalla una variable.	DUMP A;
Describe	Devuelve el schema de una variable.	DESCRIBE A;
Explain	Muestra el plan de ejecución de una relación.	EXPLAIN A;
Illustrate	Muestra una execucion paso a paso de una secuencia de sentencias.	ILLUSTRATE A;

- Operadores relacionales: Estos operadores son el corazón de Pig y conforman las principales formas de manipular los datos.

Operador	Descripcion	Sintaxis
Foreach	Realiza una transformación en cada fila.	FOREACH ... GENERATE
Filter	Permite seleccionar las tuplas que cumplen una condición.	FILTER alias BY expression
Group	Agrupa los datos en una o mas variables.	GROUP alias ALL GROUP alias BY expression GROUP alias BY (exp1, exp2, ...)
Join	Permite cruzar dos datos según campos comunes.	JOIN smaller_alias BY expression, larger_alias BY expression
Limit	Limita una cantidad el numero de tuplas de una variable.	LIMIT alias n
Order	Ordena una variable según uno o mas campos.	ORDER alias BY col [ASC|DESC]
Split	Separa una variable en dos o mas según determinadas condiciones.	SPLIT alias INTO alias1 IF expression, alias2 IF expression...
Union	Computa la unión de dos o mas relaciones.	UNION alias1, alias2
Flatten	Desanida los tipos bag y tuple.	...FLATTEN(tuple) ...FLATTEN(bag)

Para estos operadores Pig no es Case Sensitive.

- Funciones matemáticas: Funciones que realizan alguna operación matemática. Son muy similares a las que se pueden encontrar en java.lang.Math-basic. No son demasiado utilizadas.

- Funciones de evaluación: Contiene funciones útiles para el análisis de datos. En este caso Pig si es Case Sensitive.

Operador	Tipo que devuelve	Descripción	Sintaxis
AVG	double	Computa la media de los elementos de una bag.	AVG(col)
Concat	String, byte[]	Concatena dos expresiones del mismo tipo.	CONCAT(String expression1, String expression2) CONCAT(byte[] expression1, byte[] expression2)
Count	Long	Computa el numero de elementos de una bolsa.	COUNT(DataBag bag)
IsEmpty	boolean	Comprueba si un bag o map están vacios.	IsEmpty(DataBag bag), IsEmpty(Map map)
Max	Int, float, double, long	Computa el máximo valor numérico de una bolsa.	MAX(col)
Sum	Double, long	Computa la suma de los valores numéricos de un bag.	SUM(col)
Tokenize	DataBag	Separa un String y devuelve una bolsa de palabras.	TOKENIZE(String expression [, 'field_delimiter'])

- Funciones de String: Proporciona distintas funciones para analizar textos. Son comúnmente utilizadas en otros lenguajes.

Operador	Tipo que devuelve	Descripción	Sintaxis
EqualsIgnoreCase	boolean	Comprueba si el primer argumento termina con el string del segundo.	EqualsIgnoreCase(String string1, String string2)
INDEXOF	Int	Devuelve el índice de la primera ocurrencia del carácter en un String.	INDEXOF(String string, String 'character', int startIndex)

Lower	String	Convierte todos los caracteres a minúscula.	LOWER(String expression)
Ltrim	String	Devuelve una copia del String de entrada sin espacios en blanco.	LTRIM(String expression)
Replace	String	Sustituye el carácter existente en un String por el nuevo.	REPLACE(String string, String 'regExp', String 'newChar')
Upper	String	Devuelve una copia del String en mayúsculas.	UPPER(String expression)

- Funciones para Datetime: Se trata de funciones que permiten analizar fechas.

Operador	Tipo que devuelve	Descripción	Sintaxis
CurrentTime	DateTime	Devuelve un objeto DateTime con la fecha actual.	CurrentTime()
GetDay	int	Devuelve el día de la fecha.	GetDay(DateTime datetime)
GetMinute	int	Devuelve el minuto de la fecha.	GetMinute(DateTime datetime)
ToString	String	Convierte a String un objeto DateTime.	ToString(DateTime datetime)
ToDate	DateTime	Devuelve un objeto DateTime de acuerdo a unos parámetros.	ToDate(String userstring, String format, String timezone)

Las anteriores funciones presentadas son únicamente una pequeña presentación de todas las disponibles en Pig. Se puede acceder a un resumen de todos los operadores y funciones Pig mediante una tabla realizada por la empresa Mortar Data [Ref17, WWW].

Instalación y ejecución.

La instalación de Pig es muy sencilla, primero se necesita descargar una versión de la página oficial [Ref18, WWW]. Para realizar el presente trabajo fin de grado se ha utilizado la última versión disponible actualmente Apache Pig 0.13.0.

Tras descargar el archivo, se descomprime y se abre el siguiente directorio:

```
hduser@hduser-VirtualBox:~$ cd /home/hduser/Downloads
hduser@hduser-VirtualBox:~/Downloads$ sudo tar xzf pig-
0.13.0.tar.gz
[sudo] password for hduser:
hduser@hduser-VirtualBox:~/Downloads$ cd /usr/local
```

Se añade al grupo hadoop creado anteriormente pig.

```
hduser@hduser-VirtualBox:/usr/local$ sudo mv
/home/hduser/Downloads/pig-0.13.0 pig
hduser@hduser-VirtualBox:/usr/local$ sudo chown -R hduser:hadoop
pig
hduser@hduser-VirtualBox:/usr/local$
```

Hasta este punto Pig estaría en funcionamiento, como se muestra en la figura 69:

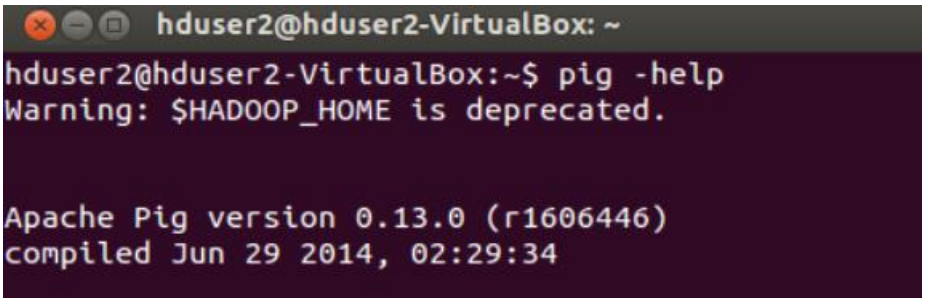


Figura 69: Pig 0.13.0 instalado.

Al igual que Hadoop proporciona comandos para su uso, Pig ofrece sus propios comandos. En general no son muy utilizados ya que mediante los comandos de Hadoop se realiza la mayor parte del trabajo. Para ejecutar un script Pig tenemos dos posibles modos:

- Modo local: En este modo se necesita acceso a una única máquina. Todos los ficheros están instalados localmente. Se especifica este modo mediante el comando pig –x local <script.pig>.

- Modo MapReduce: Para ejecutar un script Pig en este modo se necesita acceso a un clúster (o una única computadora) Hadoop y al HDFS. Este es el modo por defecto y solo necesitamos indicar el flag –x en el comando: pig –x <script.pig>. También se puede especificar el modo mediante el comando pig –x mapreduce <script .pig>.

Desde la perspectiva de un usuario no existe diferencia apreciable entre ambos modos, el modo local es utilizado para desarrollo principalmente. En este trabajo se utiliza el modo MapReduce.

3.6. Desarrollo de dos programas con Java y Pig.

Durante este apartado se van a realizar dos ejemplos con el objetivo de aplicar de forma práctica todos los conceptos vistos con anterioridad. Estos ejemplos se van a realizar cada uno en dos tecnologías: Apache Pig y JAVA, con el objetivo de realizar una comparación entre las dos posibles plataformas para realizar programas en Apache Hadoop.

3.6.1 Ejemplo 1.

En el primer ejemplo se realiza un contador de palabras. Este programa recibirá como entrada un archivo de texto almacenado en el HDFS, será capaz de separar todas las palabras y contar el número de veces que se repite cualquier término. Aunque pueda parecer un programa sin aplicación práctica, en ocasiones un contador de palabras es el comienzo de otros programas más complejos, y por tanto, es frecuentemente utilizado.

El código JAVA utilizado para este ejemplo es el siguiente:

```
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
```

```
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
public class WordCount {
      public static void main(String[] args)
                throws Exception
              {
                Configuration conf = new Configuration();

                Job job = new Job(conf, "WordCount");
                job.setJarByClass(WordCount.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(WordCount.Map.class);
                job.setReducerClass(WordCount.Reduce.class);


                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.waitForCompletion(true);
              }

             public static class Map extends Mapper<LongWritable,
Text, Text, IntWritable>
              {
                private static final IntWritable uno = new
IntWritable(1);
                private Text palabra = new Text();

                public void map(LongWritable key, Text value,
Mapper<LongWritable, Text, Text, IntWritable>.Context context) throws
IOException, InterruptedException {
                  String linea = value.toString();

                  linea = linea.toLowerCase();
                  linea = linea.replaceAll("'", "");
      linea = linea.replaceAll("[^a-zA-Z]", " ");
                  StringTokenizer tokenizer = new
StringTokenizer(linea);
                  while (tokenizer.hasMoreTokens()) {
                    this.palabra.set(tokenizer.nextToken());
                    context.write(this.palabra, uno);
                  }
                }
              }

             public static class Reduce extends Reducer<Text,
IntWritable, Text, IntWritable> {
                public void reduce(Text key, Iterable<IntWritable>
values, Reducer<Text, IntWritable, Text, IntWritable>.Context context)
throws IOException, InterruptedException {
                  int sum = 0;
                  for (IntWritable val : values) {
                    sum += val.get();
```

```
                }
                context.write(key, new IntWritable(sum));
            }
        }
}
```

Está compuesto por una clase principal y las dos clases propias del paradigma MapReduce: Map y Reduce, como se ha explicado en capítulos anteriores.

En la clase WordCount (clase principal) se realiza toda la configuración necesaria para el correcto funcionamiento del Job. Como se puede observar se especifica la clase principal, el tipo de la pareja clave/valor de salida y las clases map y reduce. Además se establece como directorio de entrada args[0] y como salida args[1] lo que implica que se debe especificar cuando se ejecuta el programa. La estructura de esta clase es común para todos los programas escritos en JAVA para Hadoop, por tanto, en el segundo ejemplo únicamente cambiará el nombre de la clase principal y los tipos de salida.

La clase map contiene un método que recibe como entrada parejas clave/valor. Este método se encarga de separar las palabras de cada línea e ir añadiéndolas como salida a la etapa Reduce con valor igual a 1. Como se explicó en apartados anteriores, el valor por defecto de dichas parejas es la línea y el número de palabra dentro del archivo de texto de entrada. El primer paso es obtener las líneas de texto (pasando los valores a texto) y eliminar caracteres especiales utilizando una expresión regular propia del lenguaje JAVA (se sustituye por espacion en blanco todo aquello que no es una letra). Tras este paso, se utliza la clase Tokenizer para ir extrayendo las palabras de cada línea. Finalmente, cada palabra (clave) se pasa como salida junto con su valor de 1.

En la clase Reduce, se recibe como entrada cada palabra como clave y con un valor de 1 cada una. En el caso de palabras con más de una ocurrencia se recibe una lista de 1 como valor debido a la operación sort implícita en hadoop. Durante este proceso se reduce esta lista sumando todos los valores 1 y definiendo así el parámetro valor. Es decir, el bucle for se encarga de realizar la suma de los valores para aquellas palabras con más de una ocurrencia.

Para poder ejecutar el anterior código se necesita crear un archivo JAVA (.jar). Este tipo de archivo permite ejecutar aplicaciones escritas en JAVA. Al estar utilizando clases propias de Hadoop es necesario añadir estas para poder compilar y después exportar el archivo .jar. Para ello se necesita el archivo hadoop-core.jar:

hadoop-
core-1.2.1.jar

Figura 70: Archivo .jar necesario.

Este archivo se puede encontrar en cualquiera de las versiones de Hadoop una vez se ha descargado. Para asociarlo a un proyecto en Eclipse IDE se realiza lo siguiente:

Botón derecho sobre el proyecto proyecto>properties>Java built path>Add External JAR´s (ver Figura 71).

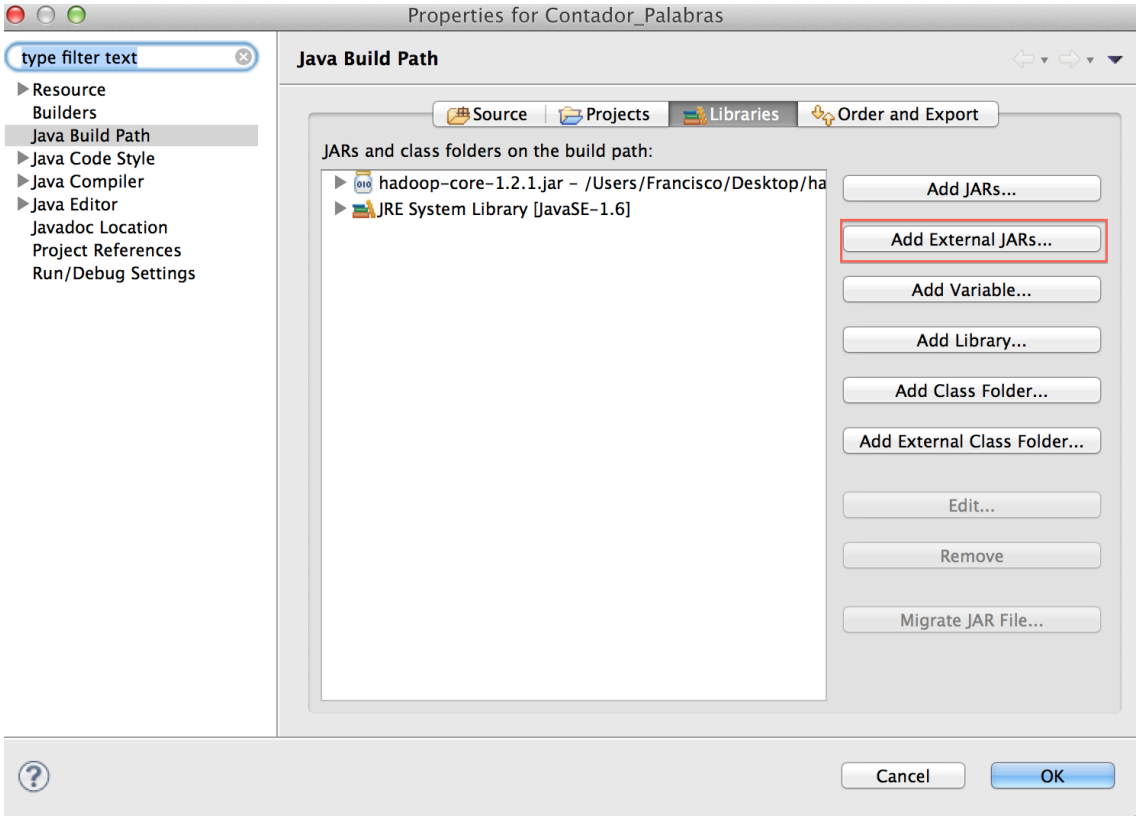


Figura 71: Añadir .jar a un proyecto JAVA.

Después se debe ejecutar el proyecto como aplicación JAVA (saldrán errores o warnings). Posteriormente, podemos exportar el archivo .jar. El proceso es el siguiente: botón derecho>export>JAR file y elegimos la ruta donde se guarda. En este caso se elige el nombre contador.jar.

Ahora se puede ejecutar el programa en Hadoop. Para ello se va a utilizar el siguiente archivo de texto de la figura 72:

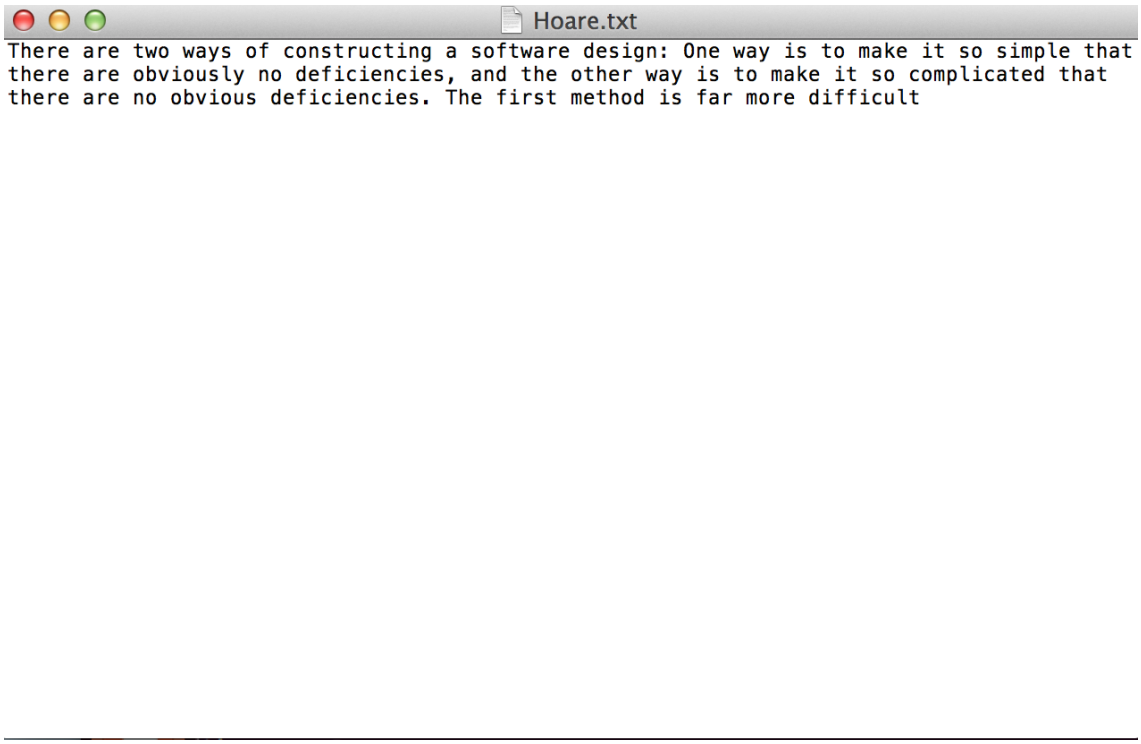


Figura 72: Archivo ejemplo 1.

Si nos fijamos en la primera palabra “there”, vemos que esta repetida en tres ocasiones, en los resultados se comprobará si esto se cumple.

Para poder utilizar el archivo, necesita estar en el HDFS:

```
hduser@hduser-VirtualBox:~/Downloads$ hadoop fs -put Hoare.txt /user/hduser/ejem
plo_contador/Hoare.txt
Warning: $HADOOP_HOME is deprecated.
```

Figura 73: Comando put en Hadoop.

Finalmente, se puede ejecutar el programa:

```
hduser@hduser-VirtualBox:~/Downloads$ hadoop jar Contador.jar
WordCount /user/hduser/ejemplo_contador/Hoare.txt
/user/hduser/ejemplo_contador/resultado
Warning: $HADOOP_HOME is deprecated.

14/08/04 14:35:52 WARN mapred.JobClient: Use GenericOptionsParser for
parsing the arguments. Applications should implement Tool for the
same.
14/08/04 14:35:52 INFO input.FileInputFormat: Total input paths to
process : 1
14/08/04 14:35:52 INFO util.NativeCodeLoader: Loaded the native-hadoop
library
14/08/04 14:35:52 WARN snappy.LoadSnappy: Snappy native library not
loaded
```

```
14/08/04 14:35:53 INFO mapred.JobClient: Running job:
job_201408041415_0002
14/08/04 14:35:54 INFO mapred.JobClient:  map 0% reduce 0%
14/08/04 14:35:59 INFO mapred.JobClient:  map 100% reduce 0%
14/08/04 14:36:07 INFO mapred.JobClient:  map 100% reduce 33%
14/08/04 14:36:09 INFO mapred.JobClient:  map 100% reduce 100%
14/08/04 14:36:10 INFO mapred.JobClient: Job complete:
job_201408041415_0002
14/08/04 14:36:10 INFO mapred.JobClient: Counters: 29
14/08/04 14:36:10 INFO mapred.JobClient:   Job Counters
14/08/04 14:36:10 INFO mapred.JobClient:     Launched reduce tasks=1
14/08/04 14:36:10 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=6012
14/08/04 14:36:10 INFO mapred.JobClient:     Total time spent by all
reduces waiting after reserving slots (ms)=0
14/08/04 14:36:10 INFO mapred.JobClient:     Total time spent by all
maps waiting after reserving slots (ms)=0
14/08/04 14:36:10 INFO mapred.JobClient:     Launched map tasks=1
14/08/04 14:36:10 INFO mapred.JobClient:     Data-local map tasks=1
14/08/04 14:36:10 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=9315
14/08/04 14:36:10 INFO mapred.JobClient:   File Output Format Counters
14/08/04 14:36:10 INFO mapred.JobClient:     Bytes Written=244
14/08/04 14:36:10 INFO mapred.JobClient:   FileSystemCounters
14/08/04 14:36:10 INFO mapred.JobClient:     FILE_BYTES_READ=533
14/08/04 14:36:10 INFO mapred.JobClient:     HDFS_BYTES_READ=379
14/08/04 14:36:10 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=109986
14/08/04 14:36:10 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=244
14/08/04 14:36:10 INFO mapred.JobClient:   File Input Format Counters
14/08/04 14:36:10 INFO mapred.JobClient:     Bytes Read=253
14/08/04 14:36:10 INFO mapred.JobClient:   Map-Reduce Framework
14/08/04 14:36:10 INFO mapred.JobClient:     Map output materialized
bytes=533
14/08/04 14:36:10 INFO mapred.JobClient:     Map input records=1
14/08/04 14:36:10 INFO mapred.JobClient:     Reduce shuffle bytes=533
14/08/04 14:36:10 INFO mapred.JobClient:     Spilled Records=92
14/08/04 14:36:10 INFO mapred.JobClient:     Map output bytes=435
14/08/04 14:36:10 INFO mapred.JobClient:     Total committed heap
usage (bytes)=222101504
14/08/04 14:36:10 INFO mapred.JobClient:     CPU time spent (ms)=980
14/08/04 14:36:10 INFO mapred.JobClient:     Combine input records=0
14/08/04 14:36:10 INFO mapred.JobClient:     SPLIT_RAW_BYTES=126
14/08/04 14:36:10 INFO mapred.JobClient:     Reduce input records=46
14/08/04 14:36:10 INFO mapred.JobClient:     Reduce input groups=31
14/08/04 14:36:10 INFO mapred.JobClient:     Combine output records=0
14/08/04 14:36:10 INFO mapred.JobClient:     Physical memory (bytes)
snapshot=270835712
14/08/04 14:36:10 INFO mapred.JobClient:     Reduce output records=31
14/08/04 14:36:10 INFO mapred.JobClient:     Virtual memory (bytes)
snapshot=1950732288
14/08/04 14:36:10 INFO mapred.JobClient:     Map output records=46
```

Se visualiza en el terminal los resultados:

```
hduser@hduser-VirtualBox:~/Downloads$ hadoop fs -cat
/user/hduser/ejemplo_contador/resultado/part-r-00000
Warning: $HADOOP_HOME is deprecated.

a     1
and   1
```

```
are   3
complicated 1
constructing      1
deficiencies      2
design      1
difficult   1
far   1
first 1
is    3
it    2
make  2
method      1
more  1
no    2
obvious     1
obviously   1
of    1
one   1
other 1
simple      1
so    2
software    1
that  2
the   2
there 3
to    2
two   1
way   2
ways  1
hduser@hduser-VirtualBox:~/Downloads$
```

Se puede comprobar que la palabra “there” esta repetida tres veces en el resultado. A continuación, se va a presentar el mismo ejemplo realizado con Apache Pig.

Para realizar el contador de palabras, se ha realizado el siguiente Script Pig:

```
lineas = LOAD './Hoare.txt' AS (linea:chararray);
aux= FOREACH lineas GENERATE LOWER(linea) AS (linea2:chararray) ;
palabras = FOREACH aux GENERATE FLATTEN(TOKENIZE(linea2)) AS palabra;
palabras_filtradas = FOREACH palabras GENERATE REPLACE(palabra,'[^a-
zA-Z]','') AS palabra;
grupo_palabra = GROUP palabras_filtradas BY palabra;
contador_palabra = FOREACH grupo_palabra GENERATE
COUNT(palabras_filtradas) AS numero_palabras, group AS palabra;
ordenados = ORDER contador_palabra BY palabra ASC;
DUMP ordenados;
```

En el Script Pig se diferencian tres fases: carga de datos, procesamiento de los datos y visualización de los resultados. En este ejemplo, se carga el archivo de texto mediante el comando LOAD. En el siguiente paso, se convierten todos los caracteres a minúscula para no tener problemas al agrupar palabras con más de una ocurrencia. En los dos siguientes pasos se separa las palabras existentes en cada línea y se eliminan caracteres especiales. La función TOKENIZE devuelve un tipo bag, por tanto, se utiliza la función

FLATTEN para desanidar este tipo y obtener cada palabra individualmente. Finalmente, se utiliza el comando GROUP para agrupar palabras que tienen más de una ocurrencia y se cuenta el número de veces que existe. Para visualizar los resultados se utiliza el comando DUMP.

Para ejecutar este Script Pig se ha optado por utilizar el servicio virtualizado de Hortonworks. Este entorno nos proporciona una interfaz gráfica además de un editor de texto lo cual lo hace propicio para ejecutar pequeños programas como dicho ejemplo.

Previamente a la ejecución del programa el archivo necesita estar almacenado en el HDFS, para ello se puede utilizar la herramienta File Browser:

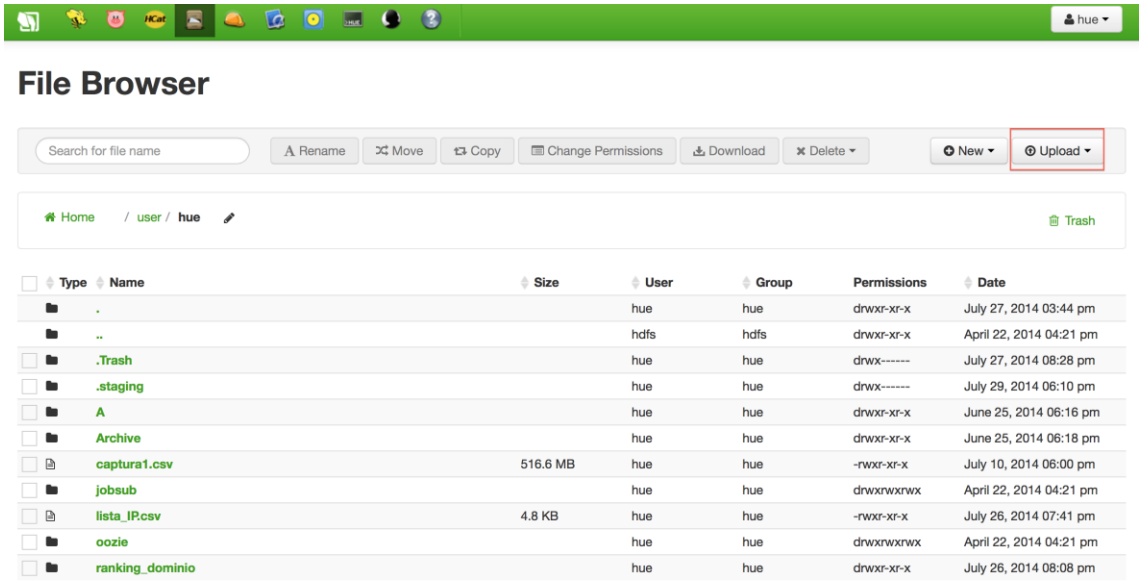


Figura 74: File Browser (HDFS).

Se selecciona la opción Upload>File y seleccionamos el archivo Hoare.txt almacenado en nuestra computadora (ver Figura 74).

La herramienta Pig permite ejecutar el Script:

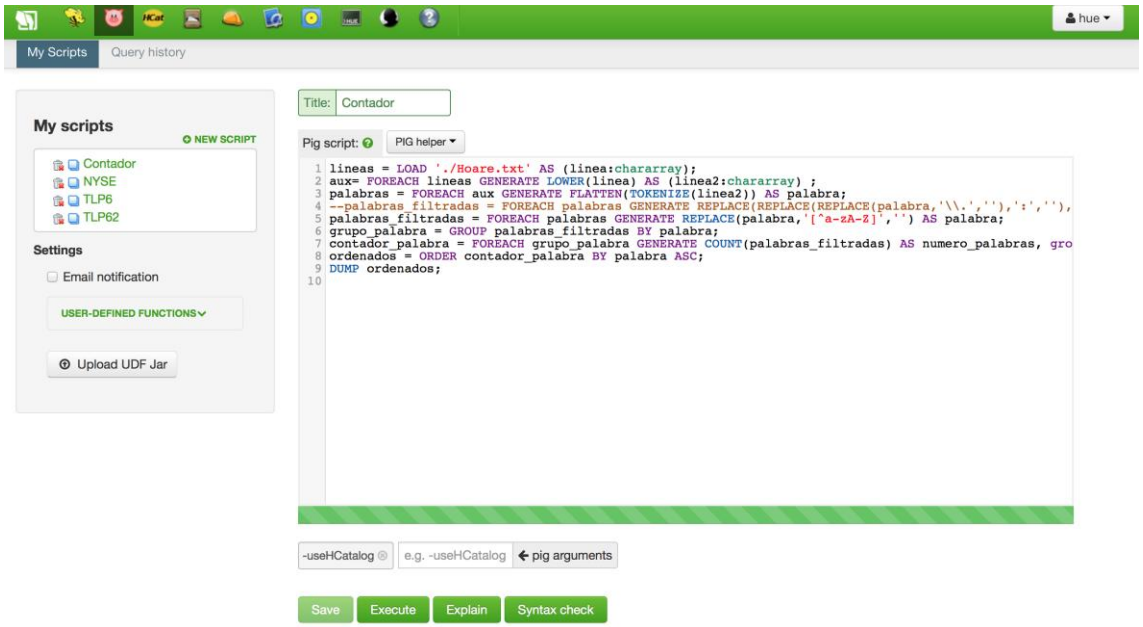


Figura 75: Ejecución completa ejemplo 1.

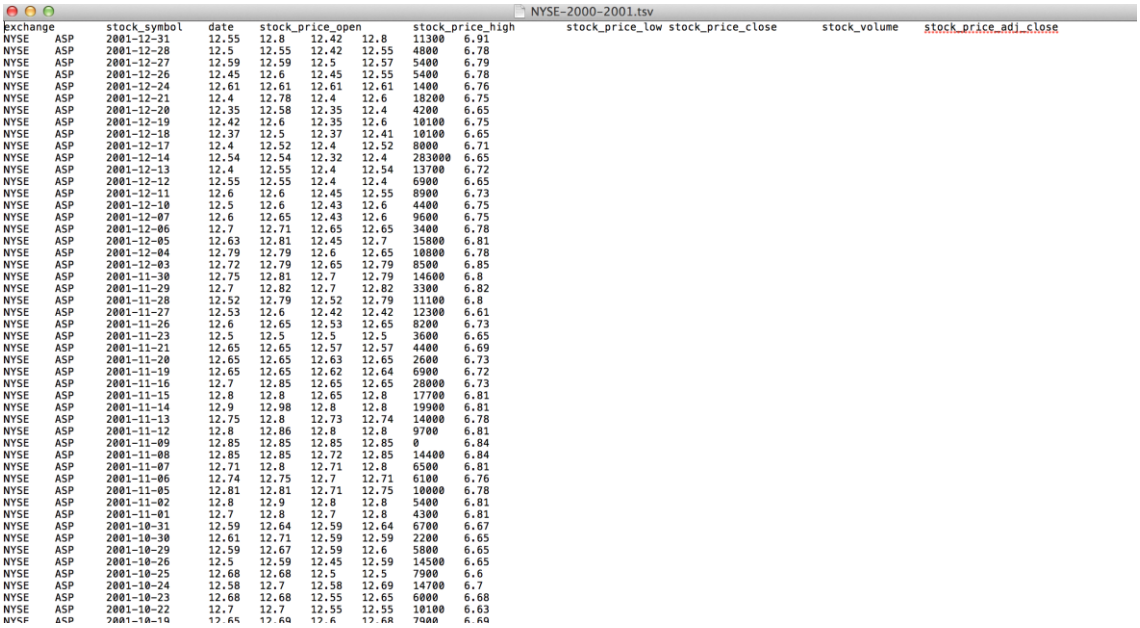
Se obtienen los siguientes resultados:

```
(1,a)
(1,and)
(3,are)
(1,complicated)
(1,constructing)
(2,deficiencies)
(1,design)
(1,difficult)
(1,far)
(1,first)
(3,is)
(2,it)
(2,make)
(1,method)
(1,more)
(2,no)
(1,obvious)
(1,obviously)
(1,of)
(1,one)
(1,other)
(1,simple)
(2,so)
(1,software)
(2,that)
(2,the)
(3,there)
(2,to)
(1,two)
(2,way)
(1,ways)
```

Como se puede comprobar son coincidentes con los resultados obtenidos con el programa escrito en JAVA.

3.6.2 Ejemplo 2.

En este segundo ejemplo se va a realizar la media del volumen de Stock de la empresa IBM entre los años 2000 y 2001. Para ello se dispone de un archivo de la bolsa de Nueva York (NYSE) con los datos de las empresas entre esos años. Una muestra del archivo es la siguiente:



NYSE-2000-2001.tsv

exchange	stock_symbol	date	stock_price_open	stock_price_high	stock_price_low	stock_price_close	stock_volume	stock_price_adj_close
NYSE	ASP	2001-12-31	12.55	12.8	12.42	12.8	11300	6.91
NYSE	ASP	2001-12-28	12.5	12.55	12.42	12.55	4800	6.78
NYSE	ASP	2001-12-27	12.59	12.59	12.5	12.57	5400	6.79
NYSE	ASP	2001-12-26	12.45	12.6	12.45	12.55	5400	6.78
NYSE	ASP	2001-12-24	12.61	12.61	12.61	12.61	1400	6.76
NYSE	ASP	2001-12-21	12.4	12.78	12.4	12.6	18200	6.75
NYSE	ASP	2001-12-20	12.35	12.58	12.35	12.4	4200	6.65
NYSE	ASP	2001-12-19	12.42	12.6	12.35	12.6	10100	6.75
NYSE	ASP	2001-12-18	12.37	12.5	12.37	12.41	10100	6.65
NYSE	ASP	2001-12-17	12.4	12.52	12.4	12.52	8000	6.71
NYSE	ASP	2001-12-14	12.54	12.54	12.32	12.4	283000	6.65
NYSE	ASP	2001-12-13	12.4	12.55	12.4	12.54	13700	6.72
NYSE	ASP	2001-12-12	12.55	12.55	12.4	12.4	6900	6.65
NYSE	ASP	2001-12-11	12.6	12.6	12.45	12.55	8900	6.73
NYSE	ASP	2001-12-10	12.5	12.6	12.43	12.6	4400	6.75
NYSE	ASP	2001-12-07	12.6	12.65	12.43	12.6	9600	6.75
NYSE	ASP	2001-12-06	12.7	12.71	12.65	12.65	3400	6.78
NYSE	ASP	2001-12-05	12.63	12.81	12.45	12.7	15800	6.81
NYSE	ASP	2001-12-04	12.79	12.79	12.6	12.65	10800	6.78
NYSE	ASP	2001-12-03	12.72	12.79	12.65	12.79	8500	6.85
NYSE	ASP	2001-11-30	12.75	12.81	12.7	12.79	14600	6.8
NYSE	ASP	2001-11-29	12.7	12.82	12.7	12.82	3300	6.82
NYSE	ASP	2001-11-28	12.52	12.79	12.52	12.79	11100	6.8
NYSE	ASP	2001-11-27	12.53	12.6	12.42	12.42	12300	6.61
NYSE	ASP	2001-11-26	12.6	12.65	12.53	12.65	8200	6.73
NYSE	ASP	2001-11-23	12.5	12.5	12.5	12.5	3600	6.65
NYSE	ASP	2001-11-21	12.65	12.65	12.57	12.57	4400	6.69
NYSE	ASP	2001-11-20	12.65	12.65	12.63	12.65	2600	6.73
NYSE	ASP	2001-11-19	12.65	12.65	12.62	12.64	6900	6.72
NYSE	ASP	2001-11-16	12.7	12.85	12.65	12.65	28000	6.73
NYSE	ASP	2001-11-15	12.8	12.8	12.65	12.8	17700	6.81
NYSE	ASP	2001-11-14	12.9	12.98	12.8	12.8	19900	6.81
NYSE	ASP	2001-11-13	12.75	12.8	12.73	12.74	14000	6.78
NYSE	ASP	2001-11-12	12.8	12.86	12.8	12.8	9700	6.81
NYSE	ASP	2001-11-09	12.85	12.85	12.85	12.85	0	6.84
NYSE	ASP	2001-11-08	12.85	12.85	12.72	12.85	14400	6.84
NYSE	ASP	2001-11-07	12.71	12.8	12.71	12.8	6500	6.81
NYSE	ASP	2001-11-06	12.74	12.75	12.7	12.71	6100	6.76
NYSE	ASP	2001-11-05	12.81	12.81	12.71	12.75	10000	6.78
NYSE	ASP	2001-11-02	12.8	12.9	12.8	12.8	5400	6.81
NYSE	ASP	2001-11-01	12.7	12.8	12.7	12.8	4300	6.81
NYSE	ASP	2001-10-31	12.59	12.64	12.59	12.64	6700	6.67
NYSE	ASP	2001-10-30	12.61	12.71	12.59	12.59	2200	6.65
NYSE	ASP	2001-10-29	12.59	12.67	12.59	12.6	5800	6.65
NYSE	ASP	2001-10-26	12.5	12.59	12.45	12.59	14500	6.65
NYSE	ASP	2001-10-25	12.68	12.68	12.5	12.5	7900	6.6
NYSE	ASP	2001-10-24	12.58	12.7	12.58	12.69	14700	6.7
NYSE	ASP	2001-10-23	12.68	12.68	12.55	12.65	6000	6.68
NYSE	ASP	2001-10-22	12.7	12.7	12.55	12.55	10100	6.63
NYSE	ASP	2001-10-19	12.65	12.69	12.6	12.68	7900	6.69

Figura 76: Archivo para ejemplo 2.

La segunda columna de los datos corresponde con las iniciales de la empresa (IBM para el ejemplo) y en la octava el volumen de stock de cada día. Sera por tanto el objetivo, seleccionar todas las filas cuya segunda columna sea IBM y realizar la media del volumen de stock (campo 8) para todas.

```
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class NYSE{

        static class NYSEMapper extends Mapper<LongWritable, Text,
Text, DoubleWritable> {
                private Text empIBM = new Text();

      public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {

      String siguienteCampo = value.toString();
          String campos[] = siguienteCampo.split("\t");
                  String empresa= "IBM";

                  if(campos[1].equals(empresa)){
                        empIBM.set(campos[1]);

                    double stock_volumen =
Double.parseDouble(campos[7]);

               context.write(empIBM, new
DoubleWritable(stock_volumen));

                }


            }
          }
        static class NYSEReducer extends Reducer<Text, DoubleWritable,
Text, DoubleWritable> {
          public void reduce(Text key, Iterable<DoubleWritable>
values, Context context) throws IOException, InterruptedException {
            double sum = 0.0;
            int count=0;
            for (DoubleWritable value: values) {
              sum += value.get();
              count++;
            }

            context.write(key, new DoubleWritable(sum/count));
          }
        }

          public static void main(String[] args) throws Exception
        {
          Job job = new Job();
          job.setJarByClass(NYSE.class);
          FileInputFormat.addInputPath(job, new Path(args[0]));
          FileOutputFormat.setOutputPath(job, new Path(args[1]));
          job.setMapperClass(NYSEMapper.class);
          job.setReducerClass(NYSEReducer.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(DoubleWritable.class);
          System.exit(job.waitForCompletion(true)? 0:1);
        }
}
```

La estructura del código es muy similar al primer ejemplo. El método main únicamente es modificado en los nombres de la clase y métodos, el resto es idéntico al primer ejemplo.

En el método map, se recibe como entrada cada fila del documento. En todas ellas, se busca el parámetro que contiene el nombre de la empresa IBM. Si se encuentra una fila cuyo campo es igual a "IBM" se pasa como salida de esta etapa junto con el campo que contiene el volumen de stock. Esto se realiza mediante un array llamado campos. Este array guarda en cada iteración la línea correspondiente.

El método reduce, recibe como entrada una una clave (IBM) y una lista de valores correspondientes a los valores del volumen de stock de cada día. Esto es proveniente del proceso de sort y shuffle implícito en Hadoop. En este proceso se realiza la media de la lista de valores. Para ello se utiliza la variable count que cuenta el número de elementos que contiene la lista de valores.

Tras escribir el código, se creará el .jar y subirá el archivo al HDFS como se ha descrito en el primer ejemplo. Se puede pasar entonces a ejecutar el archivo:

```
hduser@hduser-VirtualBox:~/Downloads$ hadoop jar NYSE.jar NYSE
/app/hadoop/tmp/NYSE-2000-2001.tsv /app/hadoop/tmp/out8
Warning: $HADOOP_HOME is deprecated.

14/07/31 14:26:22 WARN mapred.JobClient: Use GenericOptionsParser for
parsing the arguments. Applications should implement Tool for the
same.
14/07/31 14:26:22 INFO input.FileInputFormat: Total input paths to
process : 1
14/07/31 14:26:22 INFO util.NativeCodeLoader: Loaded the native-hadoop
library
14/07/31 14:26:22 WARN snappy.LoadSnappy: Snappy native library not
loaded
14/07/31 14:26:23 INFO mapred.JobClient: Running job:
job_201407311321_0007
14/07/31 14:26:24 INFO mapred.JobClient:  map 0% reduce 0%
14/07/31 14:26:31 INFO mapred.JobClient:  map 100% reduce 0%
14/07/31 14:26:39 INFO mapred.JobClient:  map 100% reduce 33%
14/07/31 14:26:41 INFO mapred.JobClient:  map 100% reduce 100%
14/07/31 14:26:42 INFO mapred.JobClient: Job complete:
job_201407311321_0007
14/07/31 14:26:42 INFO mapred.JobClient: Counters: 29
14/07/31 14:26:42 INFO mapred.JobClient:   Job Counters
14/07/31 14:26:42 INFO mapred.JobClient:     Launched reduce tasks=1
14/07/31 14:26:42 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=8050
14/07/31 14:26:42 INFO mapred.JobClient:     Total time spent by all
reduces waiting after reserving slots (ms)=0
14/07/31 14:26:42 INFO mapred.JobClient:     Total time spent by all
maps waiting after reserving slots (ms)=0
14/07/31 14:26:42 INFO mapred.JobClient:     Launched map tasks=1
14/07/31 14:26:42 INFO mapred.JobClient:     Data-local map tasks=1
14/07/31 14:26:42 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=9421
14/07/31 14:26:42 INFO mapred.JobClient:   File Output Format Counters
14/07/31 14:26:42 INFO mapred.JobClient:     Bytes Written=13
```

```
14/07/31 14:26:42 INFO mapred.JobClient:   FileSystemCounters
14/07/31 14:26:42 INFO mapred.JobClient:     FILE_BYTES_READ=7006
14/07/31 14:26:42 INFO mapred.JobClient:     HDFS_BYTES_READ=44006084
14/07/31 14:26:42 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=122966
14/07/31 14:26:42 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=13
14/07/31 14:26:42 INFO mapred.JobClient:   File Input Format Counters
14/07/31 14:26:42 INFO mapred.JobClient:     Bytes Read=44005963
14/07/31 14:26:42 INFO mapred.JobClient:   Map-Reduce Framework
14/07/31 14:26:42 INFO mapred.JobClient:     Map output materialized
bytes=7006
14/07/31 14:26:42 INFO mapred.JobClient:     Map input records=812990
14/07/31 14:26:42 INFO mapred.JobClient:     Reduce shuffle bytes=7006
14/07/31 14:26:42 INFO mapred.JobClient:     Spilled Records=1000
14/07/31 14:26:42 INFO mapred.JobClient:     Map output bytes=6000
14/07/31 14:26:42 INFO mapred.JobClient:     Total committed heap
usage (bytes)=222101504
14/07/31 14:26:42 INFO mapred.JobClient:     CPU time spent (ms)=2530
14/07/31 14:26:42 INFO mapred.JobClient:     Combine input records=0
14/07/31 14:26:42 INFO mapred.JobClient:     SPLIT_RAW_BYTES=121
14/07/31 14:26:42 INFO mapred.JobClient:     Reduce input records=500
14/07/31 14:26:42 INFO mapred.JobClient:     Reduce input groups=1
14/07/31 14:26:42 INFO mapred.JobClient:     Combine output records=0
14/07/31 14:26:42 INFO mapred.JobClient:     Physical memory (bytes)
snapshot=273854464
14/07/31 14:26:42 INFO mapred.JobClient:     Reduce output records=1
14/07/31 14:26:42 INFO mapred.JobClient:     Virtual memory (bytes)
snapshot=1950732288
14/07/31 14:26:42 INFO mapred.JobClien
```

Se obtiene el siguiente resultado:

```
hduser@hduser-VirtualBox:~/Downloads$ hadoop fs -cat
/app/hadoop/tmp/out8/part-r-00000
Warning: $HADOOP_HOME is deprecated.

IBM   7915934.0
hduser@hduser-VirtualBox:~/Downloads$
```

Para realizar este ejemplo, se ha realizado el siguiente Script Pig:

```
data = LOAD './NYSE-2000-2001.tsv' using PigStorage('\t') AS
(exchange:chararray, stock_symbol: chararray, date:chararray,
stock_price_open:double, stock_price_high:double, stock_price_low:
double, stock_price_close:double, stock_volume:int,
stock_price_adj_close: double);
b = filter data by stock_symbol == 'IBM';
c = group b all;
d = foreach c generate AVG(b.stock_volume);
dump d;
```

En este Script se sigue la misma estructura que en el primer ejemplo. Tras cargar los datos, se utiliza el operador FILTER para seleccionar solo aquellas entradas cuyo campo stock_symbol es IBM. De este modo solo seleccionamos las entradas de la empresa IBM. Posteriormente, se realiza la media del campo volumen de stock para dichas

entradas. Para ello se utiliza la instrucción GROUP (alias) ALL que crea un tipo bag que agrupa los datos según todos los campos.

Tras ejecutar el script, se obtiene el siguiente resultado:

```
(7915934.0)
```

Se puede comprobar como es coincidente con el resultado obtenido con el código escrito en JAVA.

3.6.3 ¿Por qué elegir Pig?

Como se puede constatar si comparamos los ejemplos en JAVA con los escritos en Pig, estos últimos resultan mucho mas concisos y con un nivel de abstracción muy superior. Pig facilita enormemente el desarrollo de programas para Hadoop ya que brinda un lenguaje de alto nivel sencillo y fácilmente entendible. La funcionalidad de los operadores Pig es fácilmente reconocible, sin embargo, los programas escritos en JAVA requieren de un mayor esfuerzo para poder ser entendidos. Para realizar cualquier programa en Pig se necesitan menos del 50% de las líneas necesarias para escribir dicho programa en JAVA. Por tanto, el tiempo necesario para crear los programas en Pig es considerablemente menor. En contra, los programas escritos en JAVA requieren menos tiempo de computación y aprovechan toda la funcionalidad que proporciona Hadoop. A pesar de ello el optimizador de Pig sigue mejorando y es probable que en un corto periodo de tiempo esta desventaja sea menos pronunciada.

Puede ocurrir en aplicaciones muy específicas que la programación se desarrolle en JAVA ya que en Pig puede ser difícilmente expresable. La creación de Scripts Pig demasiado complejos puede derivar en Jobs que requieren un gran tiempo de computación.

Una de las principales ventajas de Pig es su facilidad de uso. Su alto nivel de abstracción hace de Pig una herramienta fácil de aprender y utilizar. Por tanto, permite al desarrollador centrarse en analizar los datos en lugar de cómo analizarlos. Además permite que un usuario si amplios conocimientos de programación pueda utilizar Hadoop.

Otra ventaja de Pig es que su código es fácil de mantener. Al tratarse de un lenguaje tan conciso hacer grandes modificaciones supone cambiar una única línea. Sin embargo, realizar modificaciones en un código JAVA MapReduce puede ser bastante complejo.

Todas estas ventajas provocan que Pig sea la herramienta mas adecuada para el análisis de datos en Hadoop y garantizan su uso y desarrollo en el futuro. Es por todo ello que se ha decidido utilizar esta herramienta para analizar los datos provenientes de la Teleco LAN party 6 y que será el tema principal en el siguiente capítulo.

Capítulo 4. Análisis de los datos de la TLP6.

Durante este capítulo se va a analizar los datos de tráfico generados durante un día por la Teleco LAN Party Cartagena (ver Figura 77) celebrada en la biblioteca de la Universidad Politécnica de Cartagena entre los días 10 y 15 de Abril y con un total de 300 usuarios. El tráfico a analizar será el generado en un día, en concreto, durante el día 11 de Abril que coincide con el primer día completo de dicho evento. Los datos de tráfico correspondientes al tráfico LAN se recogieron mediante el analizador de protocolos WireShark. Como se ha comentado en capítulos previos, el lenguaje de programación utilizado en el análisis será Apache Pig soportado por el framework Hadoop.

Figura 77: Logo Teleco LAN Party.

Durante el evento los usuarios disfrutaron de una velocidad de descarga de 1 Gbps, que provocó una ingente cantidad de tráfico durante el acontecimiento. Además se disponía de una red WIFI de alta velocidad mas no se capturo el tráfico de la misma.

4.1 Exploración de los datos y objetivos.

En el trascurso del día 11 de Abril se generaron alrededor de 150 gigabits de datos que están repartidos en 13 carpetas de aproximadamente 12 GB cada uno. Estas carpetas contienen a su vez ficheros en formato .csv, los cuales recogen la captura de tráfico cada 200 segundos. Constar que las 13 carpetas se encuentran de inicio comprimidas en formato .7z. Esta estructura se muestra en las figuras 78 y 79.

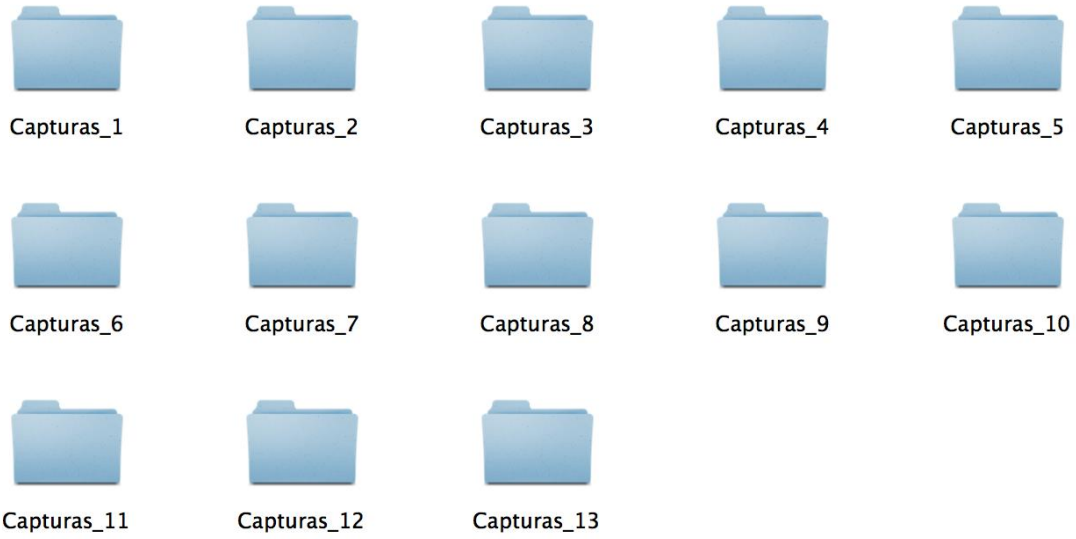


Figura 78: Estructura de carpetas de los datos.

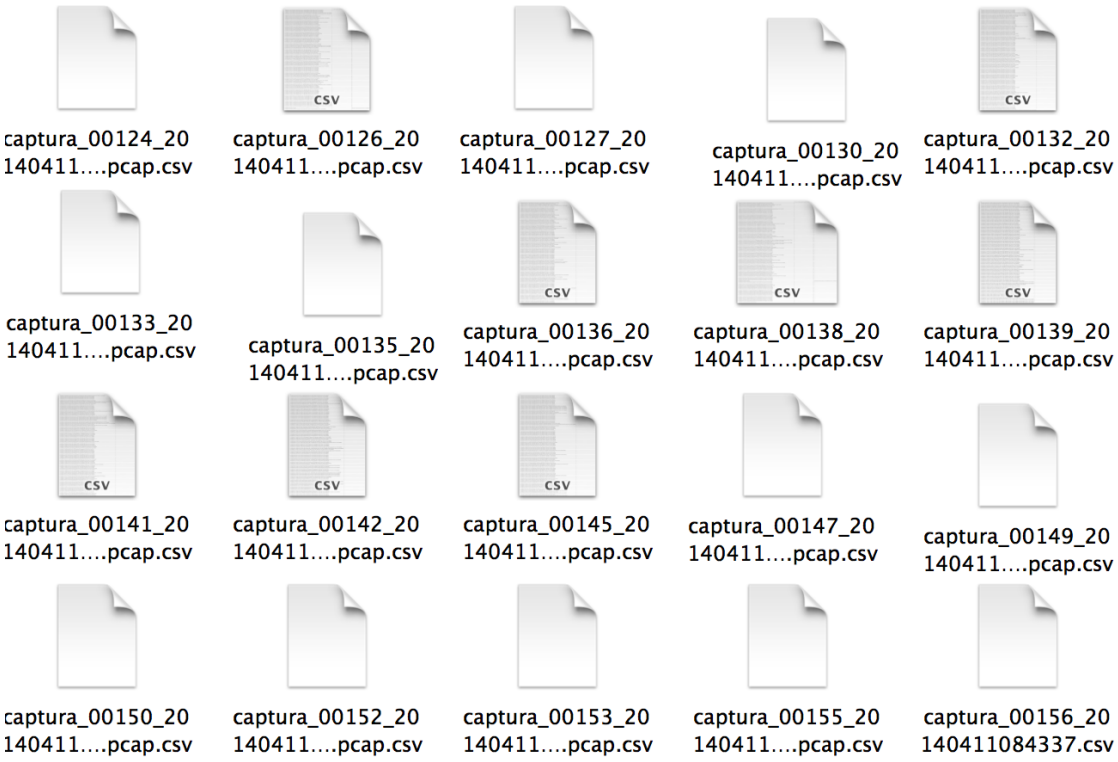


Figura 79: Interior de una carpeta.

Es importante constar que el tiempo que se va a analizar es el que transcurre entre las dos siguientes fechas: 2014-04-10 22:42:00 y 2014-04-12 01:44:00. Lo cual es , en realidad algo más de 24 horas, si tenemos en cuenta que entre las dos fechas tenemos 1623 minutos el numero de ficheros aproximado debería ser de 490. En realidad se disponen de unos 310, lo cual indica que durante la captura de los datos alguna anomalía provoco la perdida de información. A pesar de ello, se disponen de suficientes datos para realizar el análisis.

En el interior de cada fichero de tráfico se encuentra la estructura típica de campos de un analizador de protocolos. Se dispone de distintos campos de información separados por comas así como una cabecera al inicio de cada fichero que indica el nombre de cada campo. A continuación, se muestra un extracto de uno de los ficheros:

```
No.,"Time","Source","Destination","Source Port","Destination
Port","Protocol","Info","Total Time"
1,"0.000000000","217.124.208.44","80.31.193.127","50531","9635","UDP","Source port:
50531  Destination port: 9635","2014-04-11 00:06:44.868014000"
2,"0.000476000","85.53.181.70","217.124.208.78","30022","49749","TCP","30022 > 49749
[PSH, ACK] Seq=1 Ack=1 Win=64810 Len=5","2014-04-11 00:06:44.868490000"
3,"0.000865000","84.121.11.224","217.124.208.139","61861","46779","UDP","Source
port: 61861  Destination port: 46779","2014-04-11 00:06:44.868879000"
4,"0.001398000","217.124.208.170","195.57.81.57","32482","http","HTTP","Continuation
or non-HTTP traffic[Packet size limited during capture]","2014-04-11
00:06:44.869412000"
5,"0.001929000","193.126.163.96","217.124.208.78","22639","10183","TCP","22639 >
10183 [PSH, ACK] Seq=1 Ack=1 Win=49700 Len=231[Packet size limited during
capture]","2014-04-11 00:06:44.869943000"
6,"0.002979000","217.124.208.78","193.126.163.96","10183","22639","TCP","10183 >
22639 [PSH, ACK] Seq=1 Ack=232 Win=64851 Len=101","2014-04-11 00:06:44.870993000"
7,"0.003430000","217.124.208.78","109.131.28.68","49744","38186","TCP","49744 >
38186 [ACK] Seq=1 Ack=1 Win=64060 Len=0","2014-04-11 00:06:44.871444000"
```

Se puede constatar como se disponen de los siguientes 9 campos de información:

- No: Indica el numero del paquete dentro de un fichero. En el análisis mediante Pig este campo carece de valor.
- Time: Indica el tiempo que transcurre desde el incio del fichero. En el análisis global que se realiza con Pig tampoco es utilizado.
- Source: Indica la IP origen del paquete.
- Destination: Indica la IP destino del paquete.
- Source port: indica el puerto origen de la conexión.
- Destination port: indica el puerto destino de la conexión.
- Protocol: Especifica el protocolo utilizado en la conexión.
- Info: Aporta información adicional acerca del paquete. En el análisis no es utilizado ya que se dispone de suficiente información con el resto de campos.
- Total Time: Indica la fecha y hora a la que el paquete es capturado. Este podría ser uno de los campos más importantes para el análisis.

De una correcta manipulación de todos estos campos dependerá el análisis. A través de Pig se calcularan diversos parámetros para cada carpeta de ficheros. En primer lugar, se va a calcular el número total de paquetes que se intercambiaron durante el día 11 así como representaciones del número de paquetes por hora y minuto. De estos cálculos se podrá extraer las horas y minutos de mayor tráfico. Posterior al análisis del número de paquetes, se analiza el tráfico según el protocolo usado. Se representaran el número de paquetes para los protocolos más utilizados por minuto, extrayendo de estos datos además un ranking con los protocolos más usados y su porcentaje de utilización respecto del total. Por ultimo, se realizará un extenso análisis según las IP. Se calculará el número de paquetes por minuto para cada IP y se examinaran los resultados. Derivado de esto se utilizará unos datos auxiliares que contienen los dominios con las IP de las páginas más importantes. A través de este último análisis se intentará aproximar el número medio de contactos a páginas web distintas por usuario, teniendo en cuenta algunas consideraciones.

4.2 Análisis de los datos mediante Pig.

Este subapartado está dedicado a la explicación del script Pig encargado de obtener los resultados descritos anteriormente. Igualmente, se aclarará el método de ejecución de todos los datos.

Obtención de los datos.

El primer paso para el análisis de los datos viene dado por la carga de los mismos mediante el comando LOAD. En este paso se encuentra el primer problema, si se analizan los datos, se puede comprobar que el delimitador de campos son comas. Pero en algunas entradas, como por ejemplo la número 5 del extracto del tráfico mostrado en el subapartado uno, se encuentran comas dentro de los campos:

```
5,"0.001929000","193.126.163.96","217.124.208.78","22639","10183","TCP
","22639 > 10183 [PSH, ACK] Seq=1 Ack=1 Win=49700 Len=231[Packet size
limited during capture]","2014-04-11 00:06:44.869943000"
```

Se puede identificar una coma dentro del campo Info que contiene información adicional del paquete. Esto supone un problema ya que si se cargan los datos utilizando como delimitador la coma el SCHEMA no sería el mismo para todas las entradas. Para solucionar esta cuestión se decide preprocesar todos los datos con la herramienta SED [Ref19, WWW]. Se trata de una potente herramienta de tratamiento de texto para el sistema operativo Unix que acepta como entrada un archivo, lo lee y modifica línea a línea de acuerdo a un script, mostrando el resultado por salida estándar. Esto permite sustituir todas las comas que separan los campos por otro delimitador (ver Figura 80).

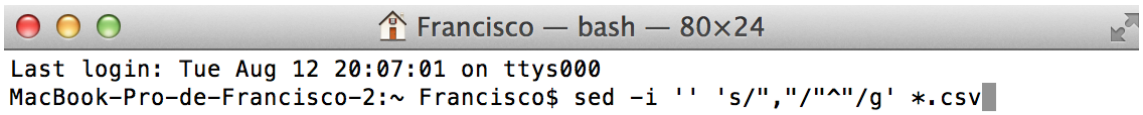


Figura 80: Comando SED.

En este caso se decide sustituir las comas por el siguiente delimitador: "^". Para realizar este reemplazo se utiliza el siguiente comando:

```
MacBook-Pro-de-Francisco-2:Archive Francisco$ sed -i '' 's/","/"^"/g'
*.csv
```

En ocasiones, para ciertos archivos, se muestra el siguiente error:

```
sed: RE error: illegal byte sequence
```

Este error tiene relación con la codificación de los caracteres, es probable que en el fichero existan caracteres no validos en UTF-8. Para solucionarlo basta con añadir al comando LC_ALL=C que varía el valor de algunos parámetros de SED:

```
MacBook-Pro-de-Francisco-2:Archive Francisco$ LC_ALL=C sed -i ''
's/","/"^"/g' *.csv
```

Es posible que este comando introduzca errores en algunos paquetes, aún así, el numero de ficheros con dicho error no es significativo.

Tras el pre procesamiento de todos los datos podemos realizar la carga de los mismos:

```
/* Obtencion de DATOS */
raw_data = LOAD './*' USING PigStorage('^');
B = FILTER raw_data BY $8 != ''; /* Eliminamos cabeceras */
/* Generamos en C los Datos con el CORRECTO ESQUEMA */
C = FOREACH B GENERATE REPLACE($0,'"','') AS numero:int,$1 AS time,
REPLACE($2,'"','') AS IPS,REPLACE($3,'"','') AS IPD,$4 AS PuertoOrigen,$5 AS
PuertoDestino, $6 AS Protocolo, $7 AS Informacion,
ToDate(RTRIM(REPLACE(REPLACE($8,'"',''),' ','T'))) AS FechayHora: datetime;
/*DESCRIBE C;*/
```

La variable raw_data carga todos los datos de un directorio. Posteriormente, se eliminan las cabeceras de todos los archivos. Estos tienen en común que su campo total time es nulo ya que el primer campo engloba a los dos primeros parámetros No. y time. Evidentemente, los dos primeros campos se perderían aunque no se necesitan para el análisis. Con la variable C se establece el esquema definitivo de los datos. Es

importante señalar la transformación del campo total time a FechayHora como tipo datetime mediante el método ToDate. Esta conversión requiere de un formato determinado, en concreto, el campo de fecha y hora han de estar separados por el carácter T. Existe un método ToDate que recibe como parámetro de entrada el formato de la fecha aunque en este caso resultaba más sencillo realizar la conversión mediante el primer modo.

Análisis de cantidad de paquetes.

Como se ha comentado se realiza un análisis de la cantidad de paquetes intercambiados así como su distribución en el tiempo.

```
/* N° total de paquetes */
group_all = GROUP C all;
total_paquetes = FOREACH group_all GENERATE COUNT(C);

/* Numero paquetes/minuto PARA REPRESENTAR */
Datos_hora_minuto = FOREACH C GENERATE
CONCAT(CONCAT((Chararray)GetHour(FechayHora),':'),(Chararray)GetMin
ute(FechayHora)) AS hora,
CONCAT(CONCAT(CONCAT((Chararray)GetYear(FechayHora),'-
'),CONCAT((Chararray)GetMonth(FechayHora),'-
')),(Chararray)GetDay(FechayHora)) AS fecha, FechayHora;
Datos_hora_minuto_group = GROUP Datos_hora_minuto BY (fecha, hora);
Representacion_paquetes_minuto = FOREACH Datos_hora_minuto_group
GENERATE FLATTEN(group), COUNT(Datos_hora_minuto) AS paquetes:int;
/*DUMP Representacion_paquetes_minuto;
DESCRIBE Representacion_paquetes_minuto;*/

/* Minutos mas y menos cargados */
por_hora = GROUP Representacion_paquetes_minuto BY hora;
horas_paquetes = FOREACH por_hora GENERATE group,
SUM(Representacion_paquetes_minuto.paquetes) AS paquetes:long;
horas_cargadas = ORDER horas_paquetes BY paquetes ASC;
/*DUMP horas_cargadas;*/
horas_menos_cargadas = ORDER horas_paquetes BY paquetes DESC;
/*DUMP horas_menos_cargadas;*/
```

El primer parámetro que se calcula es el número total de paquetes. La estructura para contar el número de entradas que contienen los datos mantiene una estructura fija en Hadoop: se agrupa la variable con todos sus campos y se cuenta el número de elementos en la bolsa (tipo bag). Otra característica interesante es una representación del número de paquetes en el tiempo. Para ello, se crea una variable que contiene el número de paquetes en cada minuto. La clave para este cálculo es la utilización de los métodos GetDay, GetMonth y GetYear para posteriormente agrupar los datos en el tiempo. Constar que para poder representar los datos convenientemente se crean los

datos con un formato determinado de fecha (separada por guiones) y hora (separada por dos puntos). Para acabar este análisis, se ordena la primera variable según orden ascendente y descendente para poder comparar los minutos con más y menos tráfico. Merece la pena comentar que la función CONCAT requiere que sus parámetros sean del tipo strings y por ello se realiza casts de los valores devueltos por la funciones Get del tipo datetime.

Posterior a este análisis y con los datos del número de paquetes por minuto, se agruparan para todas las carpetas el número de carpetas por hora para poder realizar esta representación y revisar las horas con más tráfico.

Análisis del tráfico según el protocolo.

A continuación, se realiza un análisis de los datos por protocolo. Analizando este parámetro se podrá calcular los protocolos más utilizados en las comunicaciones así como la distribución de su uso en el tiempo.

```
/* Representación Protocolo/minuto */
Datos_protocolo_minuto = FOREACH C GENERATE
CONCAT(CONCAT((Chararray)GetHour(FechayHora),':'),(Chararray)GetMinute(
FechayHora)) AS hora,
CONCAT(CONCAT(CONCAT((Chararray)GetYear(FechayHora),'-
'),CONCAT((Chararray)GetMonth(FechayHora),'-
')),(Chararray)GetDay(FechayHora)) AS fecha, FechayHora, Protocolo;
Datos_protocolo_minuto_group = GROUP Datos_protocolo_minuto BY (fecha,
hora, Protocolo);
Representacion_protocolo = FOREACH Datos_protocolo_minuto_group
GENERATE FLATTEN(group), COUNT(Datos_protocolo_minuto) AS paquetes:int;
/*DUMP Representacion_protocolo;*/

/* Ranking numero paquetes según protocolo */
Datos_protocolo = GROUP C BY Protocolo;
Ranking_protocolo = FOREACH Datos_protocolo GENERATE group, COUNT(C) AS
paquetes:long;
Ranking_protocolo_ord = ORDER Ranking_protocolo BY paquetes DESC;
/*DUMP Representacion_protocolo_ord; */

/* % Respecto del total protocolo */
Proto_group_all = GROUP Ranking_protocolo_ord ALL;
Total_paquetes = FOREACH Proto_group_all GENERATE
SUM(Ranking_protocolo_ord.paquetes) AS paquetes:long;
/*DUMP Total_paquetes;*/
Porcentaje = FOREACH Ranking_protocolo_ord GENERATE
group,paquetes,(((double)paquetes/(double)Total_paquetes.paquetes)*100)
;
/*DUMP Porcentaje;*/
```

El primer parámetro se utiliza para la posterior representación del número de paquetes en el tiempo según el protocolo utilizado. La idea es la misma que en la del análisis por paquetes añadiendo a la agrupación el parámetro protocolo. Posteriormente, se utilizan los datos para obtener una tabla de protocolos junto el numero de paquetes intercambiados. Para terminar, se averigua el porcentaje de uso de cada protocolo en relación al número total de paquetes intercambiados mediante un protocolo conocido.

Análisis según IP.

El último tipo de análisis se realiza según la dirección IP de los paquetes. Este estudio permite examinar las direcciones IP visitadas por los usuarios.

```
/* REPRESENTACION IP/minuto */
Datos_IP_minuto = FOREACH C GENERATE
CONCAT(CONCAT((Chararray)GetHour(FechayHora),':'),(Chararray)GetMinu
te(FechayHora)) AS hora,
CONCAT(CONCAT(CONCAT((Chararray)GetYear(FechayHora),'-
'),CONCAT((Chararray)GetMonth(FechayHora),'-
')),(Chararray)GetDay(FechayHora)) AS fecha, FechayHora, IPD;
Datos_IP_minuto_group = GROUP Datos_IP_minuto BY (fecha, hora, IPD);
Representacion_IP = FOREACH Datos_IP_minuto_group GENERATE
FLATTEN(group), COUNT(Datos_IP_minuto) AS paquetes:int;
/*DUMP Representacion_IP;*/

/* RANKING por IP */
Datos_IP = GROUP C BY IPD;
Ranking_IP = FOREACH Datos_IP GENERATE group, COUNT (C) AS
paquetes:long;
Ranking_IP_ord = ORDER Ranking_IP BY paquetes DESC;
/*DUMP Ranking_IP_ord;*/
```

El primer dato que se calcula permite representar el número de paquetes para cada IP por minuto. Por otra parte, se cuantifica el número de paquetes totales para cada IP y se ordena para poder comprobar cual de ellas maneja más tráfico.

Es lógico pensar que este análisis carece de sentido pues el número de IP en la red es muy elevado y la representación resultaría muy confusa. Para facilitarlo necesitamos identificar las IP de sitios web con su dominio para poder identificar así el tipo de webs visitadas. Para este proceso se crea una tabla de las webs más populares con su IP. Seguidamente se muestra un extracto de la tabla:

Google	Google.es	173.194.41.247
Google	Google.es	173.194.41.248
Facebook	Facebook.es/Facebook.com	31.13.64.64
Facebook	Facebook.es/Facebook.com	31.13.64.145

Facebook	Facebook.es/Facebook.com	172.252.110.27
Facebook	Facebook.es/Facebook.com	31.13.80.65
Twitter	Twitter.es	165.160.13.20
Twitter	Twitter.es	165.160.15.20
Tuenti	Tuenti.com / Tuenti.es	95.131.168.181
elmundo.com / www.marca.com	elmundo.com / www.marca.com	193.110.128.199
as.com	as.com	91.216.63.240
as.com	as.com	91.216.63.241
Sport.es	Sport.es	217.124.241.98

Esta tabla se cruzará con los datos según el parámetro de IP destino, es decir, se añadirá la fila con el dominio de la tabla auxiliar siempre que el paquete tenga como IP destino la IP de dicho dominio. Esto permite identificar las webs que son visitadas. Evidentemente la lista de IP es limitada y, por tanto, habrá IP que los usuarios visiten y que no sean analizados. A pesar de ello estas páginas aglutinan la mayoría de visitas en España.

```
/* REPRESENTACION DOMINIO/minuto */
lista = LOAD './lista_IP.csv'  USING PigStorage(';') AS (web:chararray,
dominio:chararray, IP:chararray, tipo:chararray, tipo2:chararray);
unido = join C by IPD, lista by IPusing 'replicated;
Datos_dominio_minuto = FOREACH unido GENERATE
CONCAT(CONCAT((Chararray)GetHour(FechayHora),':'),(Chararray)GetMinute(Fe
chayHora)) AS hora,
CONCAT(CONCAT(CONCAT((Chararray)GetYear(FechayHora),'-
'),CONCAT((Chararray)GetMonth(FechayHora),'-
')),(Chararray)GetDay(FechayHora)) AS fecha, FechayHora, IPD, web;
Datos_dominio_minuto_group = GROUP Datos_dominio_minuto BY (fecha, hora,
web);
Representacion_dominio = FOREACH Datos_dominio_minuto_group GENERATE
FLATTEN(group) , COUNT(Datos_dominio_minuto) AS paquetes:int;
/*STORE Representacion_dominio INTO './representacion_dominio3';*/
/*DUMP Representacion_dominio;*/


/* RANKING por Dominio */
Datos_dominio = GROUP unido BY web;
Ranking_dominio = FOREACH Datos_dominio GENERATE group, COUNT(unido) AS
paquetes:long;
Ranking_dominio_ord = ORDER Ranking_dominio BY paquetes DESC;
/*DUMP Ranking_dominio_ord;*/
/*STORE Ranking_dominio_ord INTO './ranking_dominio';*/

/* % respecto del total dominio */
Dom_group_all = GROUP Ranking_dominio_ord ALL;
Total_paquetes_dominio = FOREACH Dom_group_all GENERATE
SUM(Ranking_dominio_ord.paquetes) AS paquetes:long;
/*DUMP Total_paquetes;*/
Porcentaje_dom = FOREACH Ranking_dominio_ord GENERATE
group,paquetes,(((double)paquetes/(double)Total_paquetes_dominio.paquetes
)*100);
DUMP Porcentaje dom;
```

Para cruzar ambas tablas se utilizada la instrucción JOIN que permite cruzar datos según parámetros. Posterior al establecimiento del SCHEMA definitivo de los datos se calcula el parámetro Representacion_dominio que realiza una tabla con el número de paquetes por minuto para cada uno de los dominios visitados. La idea para calcular el parámetro es la misma que en los dos análisis anteriores. Tras este parámetro se calculan, como en los análisis precedentes, un ranking que identifica las webs más visitadas y el porcentaje de visitas respecto del total de visitas a todas las páginas de las que conocemos el dominio.

En el primer punto del presente capítulo se comentó la intención del cálculo de un parámetro adicional: número medio de contactos con páginas distintas por usuario. Para poder abordar este cálculo debemos identificar a los usuarios. Para ello se utilizarán las IP origen, en concreto, el rango de IP de las direcciones 217.124.208.* y 217.124.209.* que del análisis de los datos se supone como las asignadas a los participantes del evento.

Es importante tener en cuenta que la asignación de IP se realizaba con un servidor DHCP que designa las IP de forma dinámica. Las IP cedidas a los usuarios son renovadas cada cierto tiempo o cuando estos desconectan la computadora. A pesar de ello, si se supone la IP de un usuario fija durante el tiempo de tráfico que contiene cada carpeta (el método de ejecución se explica en el siguiente capítulo) el resultado puede ser bastante apropiado. Aun así, se debe tener en cuenta que el numero medio de paginas distintas visitadas se refiere a las paginas dentro de la lista de dominios, por tanto, el resultado puede considerarse como una cota inferior del resultado real pues podría aumentar por los dominios no tenidos en cuenta.

```
/* nº medio de contactos con paginas distintas por usuario */
trafico_web = FILTER unido BY (IPS MATCHES '217.124.208.*' OR IPS
MATCHES'217.124.209.*');
trafico_web_group = GROUP trafico_web BY (IPS, web);
visitas_raw = FOREACH trafico_web_group GENERATE FLATTEN(group);
visitas1 = GROUP visitas_raw BY IPS;
visitas = FOREACH visitas1 GENERATE group,COUNT(visitas_raw) AS
visitas;
/*DUMP visitas;*/
visita_media = FOREACH (GROUP visitas ALL) GENERATE
AVG(visitas.visitas);
/*DUMP visita_media;*/
```

Para seleccionar los paquetes enviados por los usuarios se filtran los datos de origen y posteriormente realizamos un doble agrupamiento para cuantificar solo un contacto a cada página distinta. El filtrado se realiza para todas las IP que coinciden con el comienzo de las IP que se asignan a los usuarios. Posteriormente se realiza la media para todos los usuarios.

4.3 Ejecución en una máquina virtual Hadoop.

La ejecución del Script Pig descrito en el subapartado anterior se ha realizado en una maquina virtual de Ubuntu con Hadoop instalado. La versión de Hadoop utilizada ha sido 1.2.1 por ser considerada como la última versión totalmente estable del software. Por otra parte, la versión de Pig utilizada ha sido la 0.13.0.

Por no disponer del hardware necesario no se ha instalado un cluster y la ejecución se ha realizado en una única máquina. El tamaño de los datos como se ha comentado asciende a alrededor de 150 GB y, por tanto, hubiera sido posible analizarlos a la vez mediante un clúster de computadoras.

Debido a limitaciones de disco duro en la máquina virtual y el número de tareas que puede realizar un único nodo se ha ejecutado el Script para cada una de las carpetas de datos en lugar de los 150 GB al mismo tiempo. Por tanto, se han realizado 13 ejecuciones para cada carpeta de un tamaño aproximado de 12 GB.

Para poder subir cada una de las carpetas al HDFS se ha utilizado un disco duro externo. Mediante el comando put se puede subir un directorio que contiene múltiples ficheros:

```
hduser2@hduser2-VirtualBox:~$ hadoop fs -put Capturas_1 /user/hduser2/lab4
```

El Script de ejecución Pig difiere en algunos puntos del explicado anteriormente. Para poder guardar los resultados se utiliza el comando Pig STORE al término del cálculo de cada uno de los parámetros. Además se debe especificar la correcta ruta del HDFS para cargar los datos de entrada. A continuación se presentan algunos cambios:

```
/* Obtencion de DATOS */
raw_data = LOAD 'lab4/*.csv' USING PigStorage('^');
r = FOREACH raw_data GENERATE $0..$7, REPLACE($8,'"','');
B = FILTER r BY $8 MATCHES '2014.*'; /* Eliminamos cabeceras */

/* Guardar datos */
STORE total_paquetes INTO 'lab4/total_paquetes';
```

Como se puede comprobar la variable B en este caso no está enfocada únicamente a eliminar las cabeceras ya que comprueba que el campo fecha tenga la estructura adecuada. Esta variación es implementada debido a que algunas entradas contenían errores en este campo y Pig no realiza la conversión a tipo datetime de ningún paquete. Este error en el formato puede haberse producido en el proceso de captura de datos o durante la preprocesación de los mismos con la herramienta SED.

El comando utilizado para ejecutar el Script Pig se explico en capítulos anteriores:

```
hduser2@hduser2-VirtualBox:~$ pig -x mapreduce TLP6.pig
```

Para obtener los resultados del HDFS se utiliza el comando get:

```
hduser2@hduser2-VirtualBox:~$ hadoop fs -get /user/hduser2/lab4/ranking_dominio
Ejecucion1
```

En la siguiente tabla se recoge el tiempo para cada una de las 13 ejecuciones:

Nº Ejecución	Tiempo (horas)
Ejecucion 1	01:39:35
Ejecucion 2	02:01:35
Ejecucion 3	01:31:40
Ejecucion 4	01:47:52
Ejecucion 5	01:26:33
Ejecucion 6	01:30:53
Ejecucion 7	01:28:00
Ejecucion 8	01:29:48
Ejecucion 9	01:33:08
Ejecucion 10	01:27:37
Ejecucion 11	01:42:52
Ejecucion 12	01:37:48
Ejecucion 13	01:50:53

En la figura 81 se representa el tiempo para cada ejecución realizada.

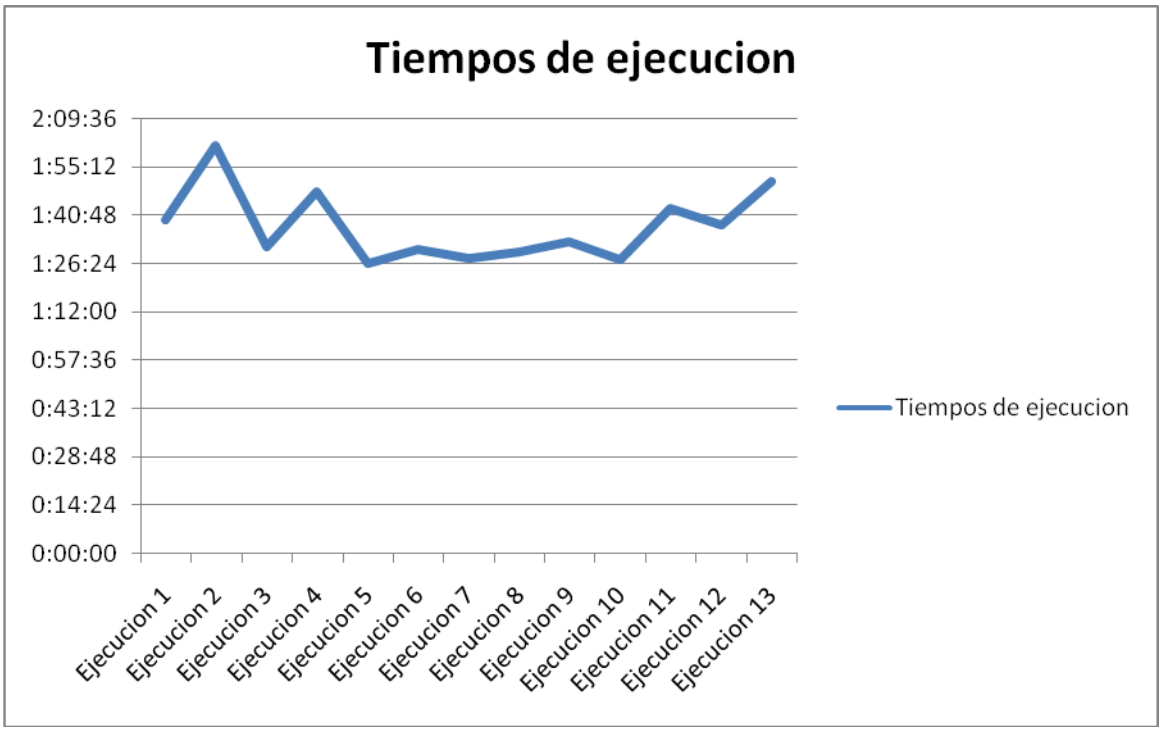


Figura 81: Tiempos de ejecución.

4.4 Resultados.

El último apartado del capítulo 4 se dedica a la presentación de resultados. El Script Pig se ha ejecutado en 13 ocasiones pero algunos de los parámetros calculados han sido reanalizados con Pig para extraer resultados globales de todas las ejecuciones.

Resultados del análisis de cantidad de paquetes.

A continuación se presentan los resultados para el primer análisis según la cantidad de paquetes.

La siguiente grafica muestra el número de paquetes por minuto de las 13 ejecuciones en conjunto. Para realizar esta grafica se ha creado un único archivo en el que se ha ido incorporando los resultados de cada ejecución:

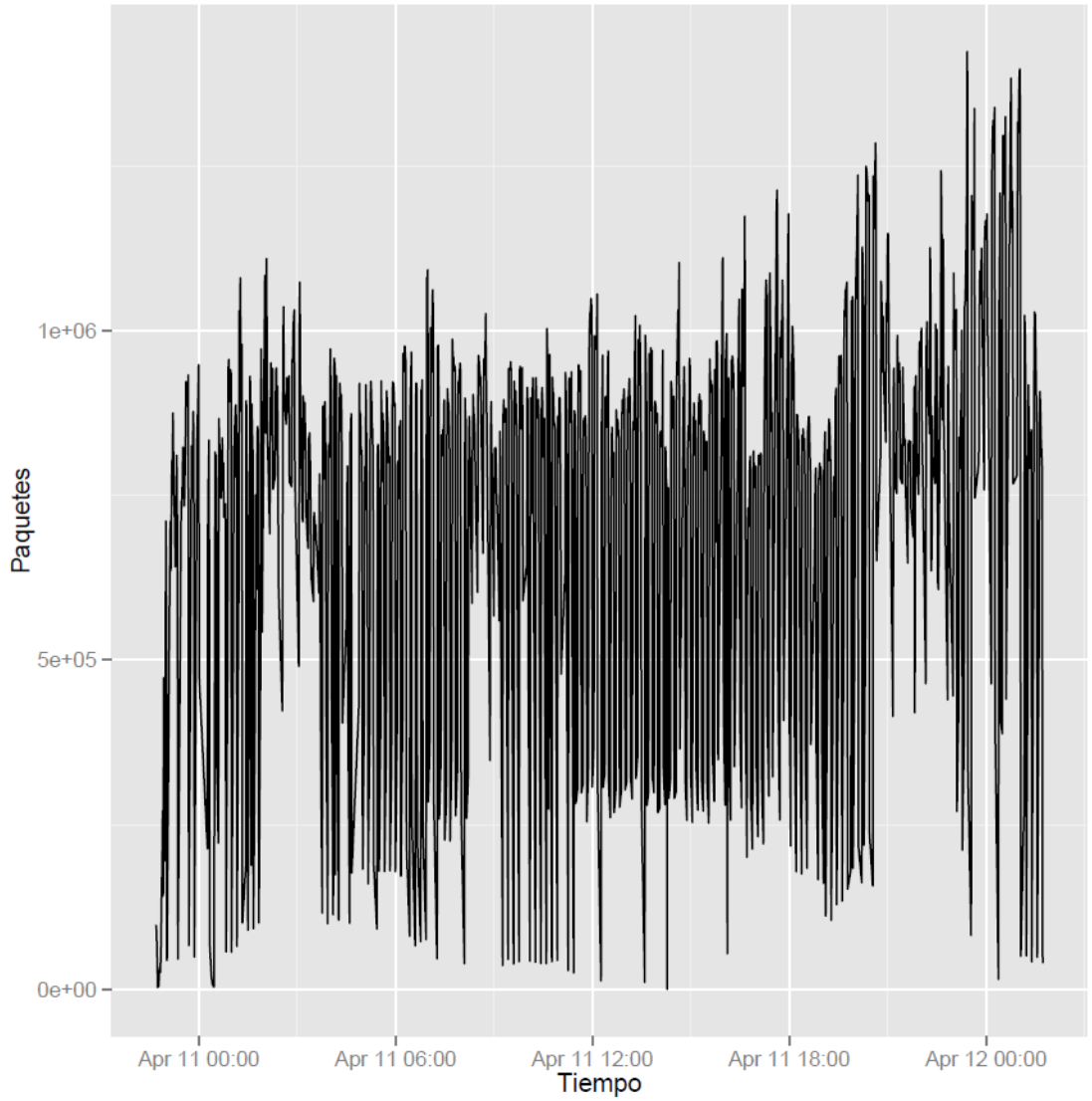


Figura 82: Numero de paquetes por minuto.

Como se puede observar la variación del número de paquetes es muy acusada provocado por la gran cantidad de tráfico durante el día. A pesar de ello, da la impresión de que el tráfico aumenta a lo largo del día produciéndose el mayor pico tráfico alrededor de la media noche.

Una de las características más notables del tráfico durante el día 11 es la escasa diferencia de tráfico entre el día y la noche. Es decir, los usuarios se mantuvieron muy activos durante las 24 horas y no descansaron durante la noche.

A continuación, se muestra los minutos de mayor tráfico para las 13 ejecuciones:

Minuto	Numero de paquetes
0:58	2256689
23:39	2251514
0:59	2235696
0:32	2107023
0:45	2106554
0:46	2099703
0:31	2096936
0:0	2095367

Como se observa los minutos de mayor tráfico se concentran alrededor de la media noche.

El resultado del número total de paquetes para cada ejecución es el siguiente:

Paquetes	Nº Ejecución
73066816	Ejecucion 1
84154470	Ejecucion 2
71821077	Ejecucion 3
61425346	Ejecucion 4
58307120	Ejecucion 5
61047073	Ejecucion 6
58971844	Ejecucion 7
60104102	Ejecucion 8
62208746	Ejecucion 9
57820940	Ejecucion 10
68411364	Ejecucion 11
63357846	Ejecucion 12
73522776	Ejecucion 13

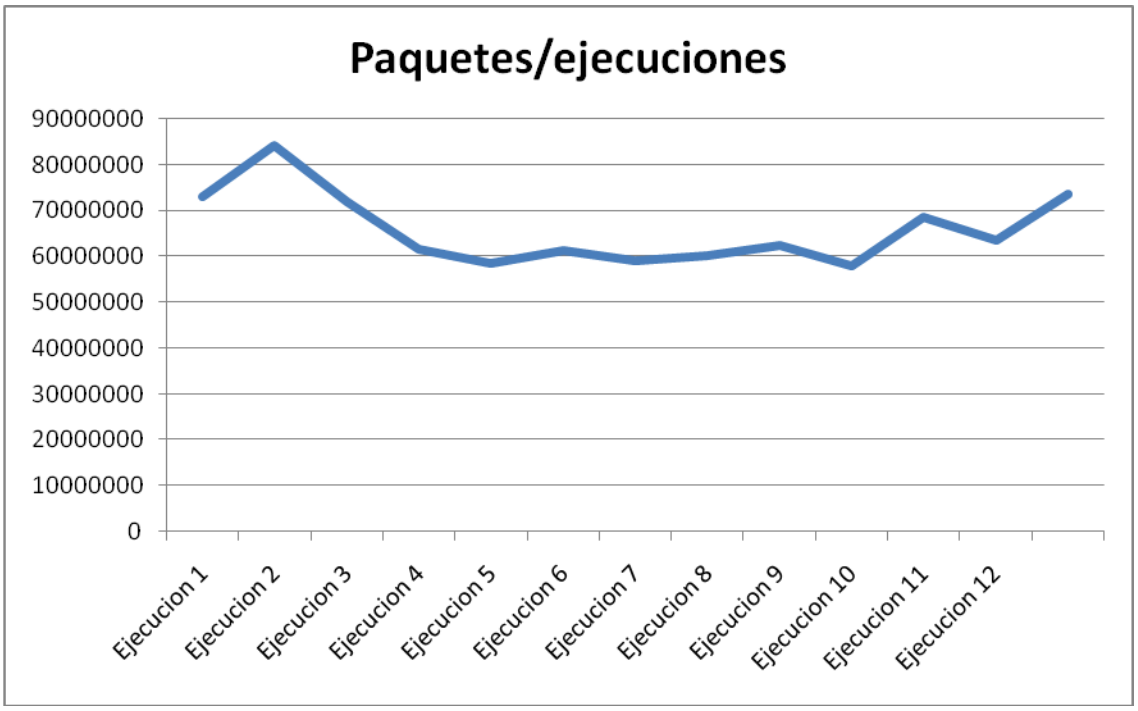


Figura 83: Numero de paquetes por ejecución.

Los resultados del número de paquetes por minuto se han reanalizado con un nuevo Script Pig para el tráfico por horas y saber así las horas con mayor intercambio de paquetes:

```
A=LOAD'./horas_cargadas/*'AS(fecha:chararray,hora:chararray,paquetes:int
);
B = FOREACH A GENERATE CONCAT((Chararray)fecha,'T') AS fecha:chararray,
hora,paquetes;
C = FOREACH B GENERATE ToDate(CONCAT( fecha, hora)) AS fecha:datetime,
paquetes;
D = GROUP C BY (GetDay(fecha), GetHour(fecha));
E = FOREACH D GENERATE FLATTEN(group), SUM(C.paquetes) AS paquetes;
ord = ORDER E BY paquetes DESC;
T = STORE E INTO './horas_cargadasresult2';
```

En el fichero horas cargadas están los 13 resultados para cada una de las ejecuciones.

Día	Hora	Paquetes
10	22	2520459
10	23	31500178
11	0	23870696
11	1	28704488

11	2	34336179
11	3	30068613
11	4	25812509
11	5	31027149
11	6	24514041
11	7	33765409
11	8	27249755
11	9	31892027
11	10	35178602
11	11	33355447
11	12	32679292
11	13	33648305
11	14	32380142
11	15	32703446
11	16	33067548
11	17	32367390
11	18	31051348
11	19	30668143
11	20	37060249
11	21	33694403
11	22	29250181
11	23	38587155
12	0	36977075
12	1	26289291

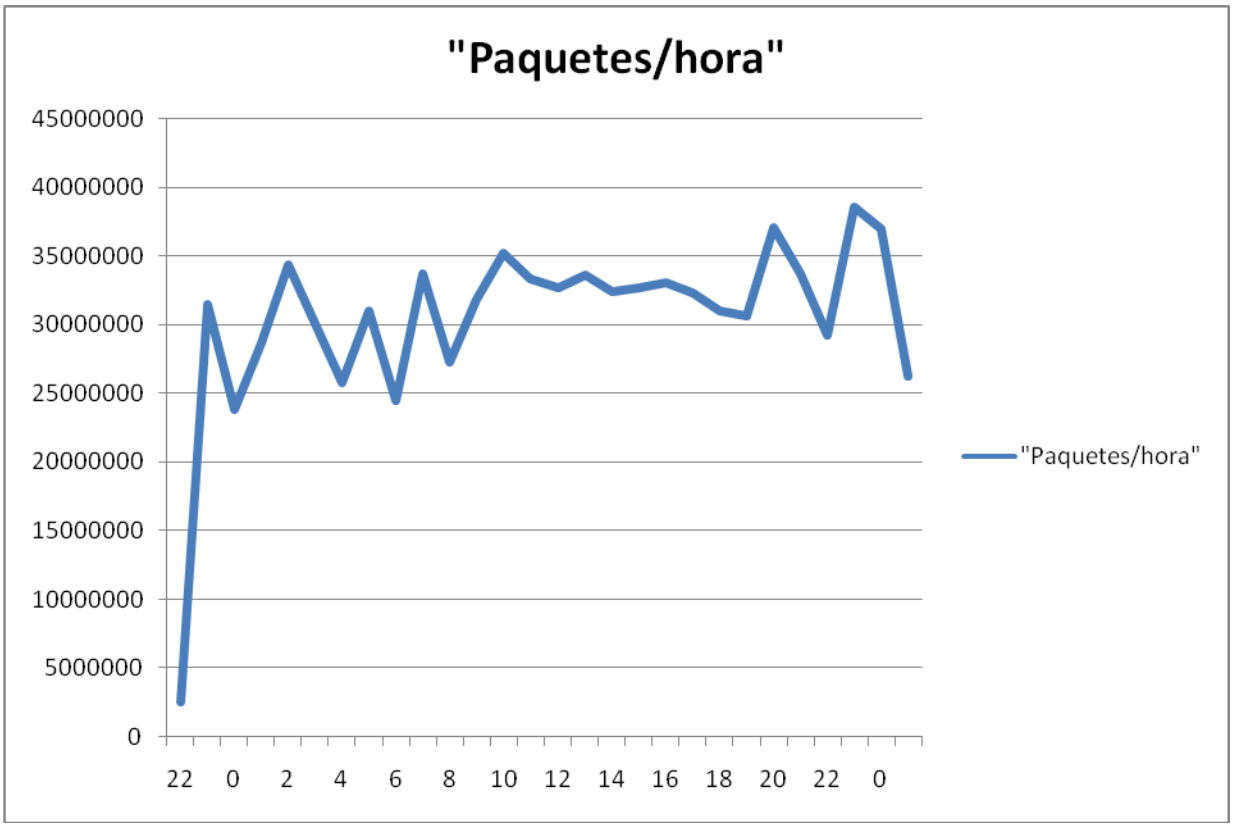


Figura 84: Paquetes por hora.

En esta gráfica se suavizan las variaciones y se puede apreciar mejor la tendencia del tráfico. La franja de descanso de los participantes parece situarse entre las 10 y las 12 de la mañana donde el tráfico desciende ligeramente.

Ahora se presenta las horas con mayor tráfico del día:

```
11    23    38587155
11    20    37060249
12    0     36977075
```

En total el número de paquetes intercambiados durante el día 11 de Abril asciende a lo siguiente:

Número total de paquetes
854219520

Resultados del análisis por protocolos.

A continuación se presentan los resultados para el análisis según el protocolo utilizado en la comunicación.

La grafica del uso de protocolos por minuto se ha realizado para las 13 ejecuciones en conjunto y solo se han representado los principales protocolos:

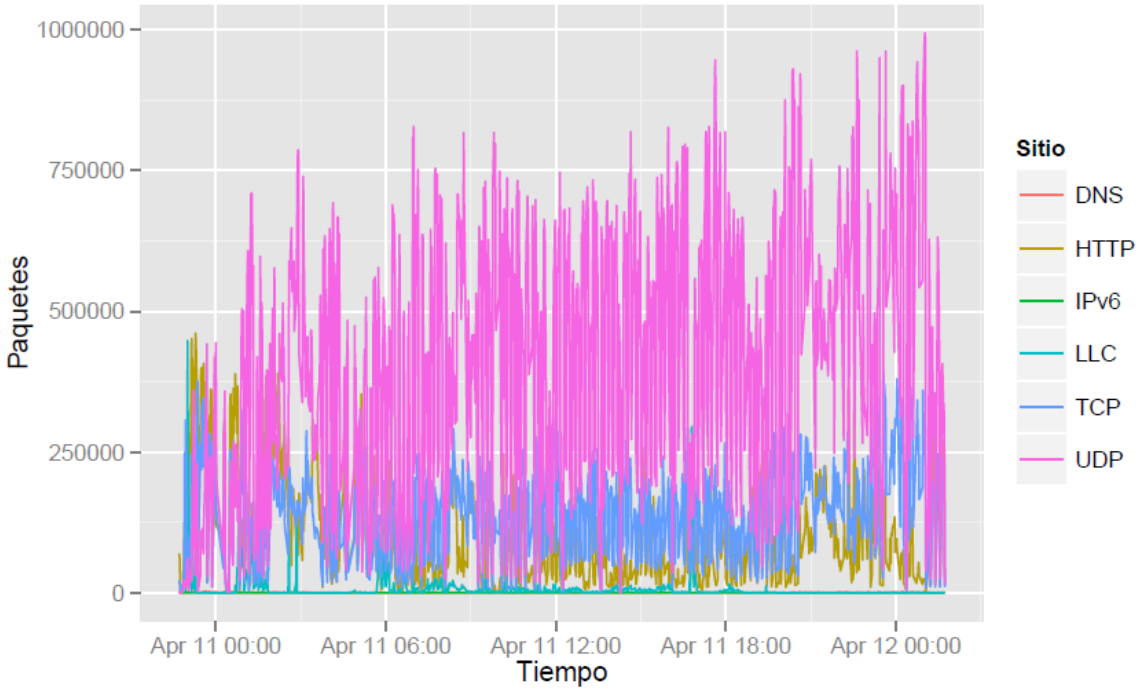


Figura 85: Numero de paquetes según protocolo por minuto.

Como se ve el uso de los protocolos es ligeramente constante. Si se representa únicamente el protocolo DNS:

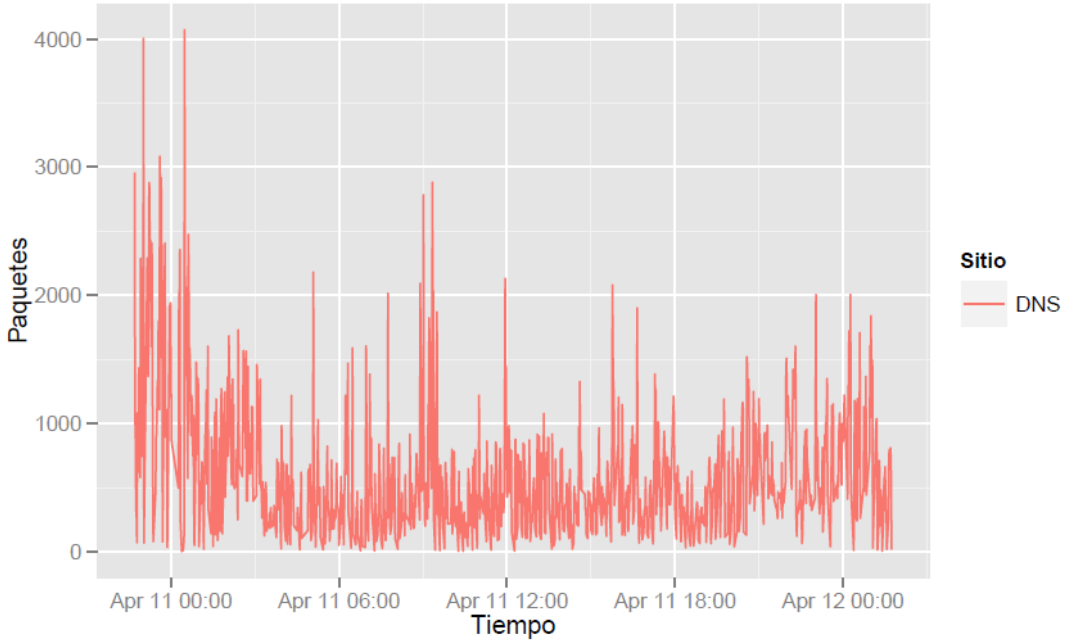


Figura 86: Trafico para el protocolo DNS.

En este caso este protocolo es más utilizado al comienzo del día. Esto ocurre porque al comienzo los usuarios no han conectado a ninguna web. Cuando estas conexiones se van produciendo las DNS se almacenan en la memoria caché y el uso del protocolo disminuye.

Para poder conocer los protocolos más utilizados durante el dia se reanalizan con el siguiente Script los resultados de las ejecuciones:

```
A = LOAD './minutos_cargados/*' AS (protocolo:chararray,paquetes:int);
B = GROUP A BY protocolo;
C = FOREACH B GENERATE group, SUM(A.paquetes) AS paquetes;
D = ORDER C BY paquetes DESC;
E = LIMIT D 8;
T = STORE E INTO './ranking_protocolo_cargados';
```

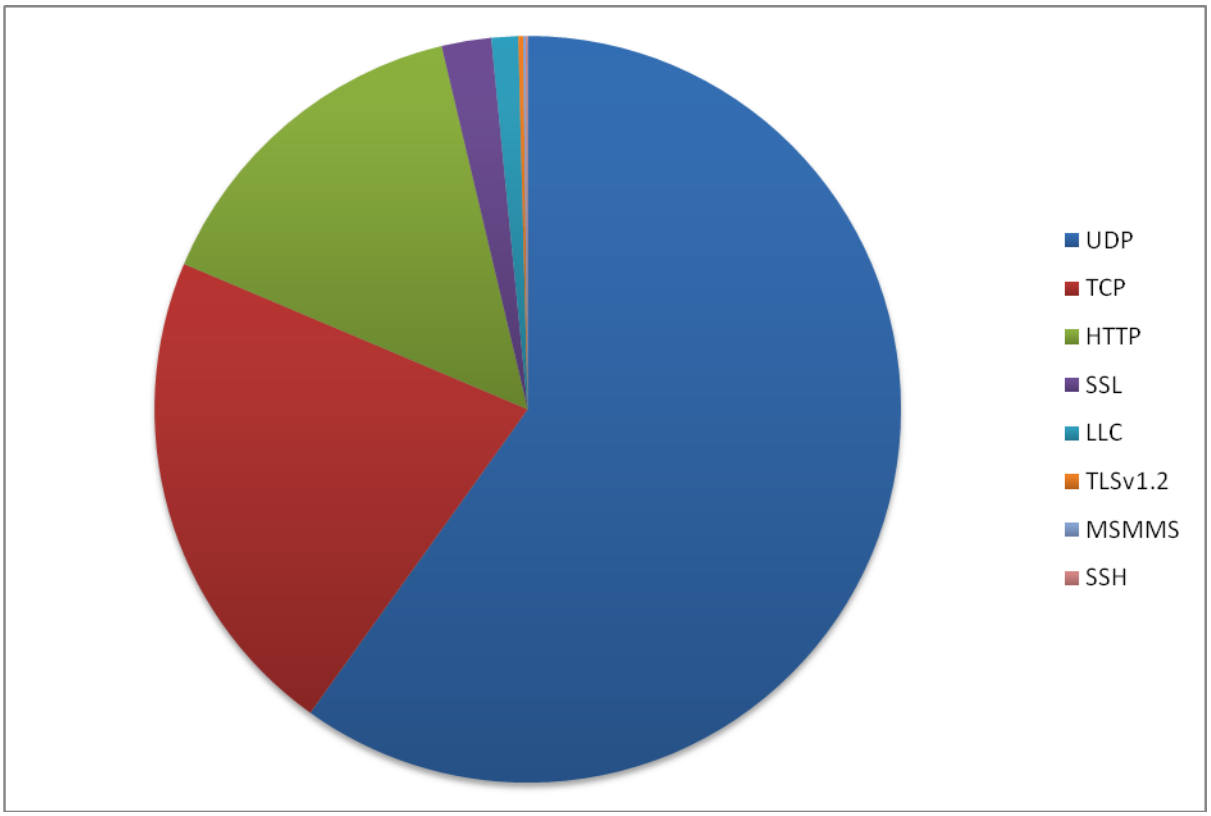
Protocolo	Nºtotal de paquetes	Porcentaje (%)
UDP	509440190	59,90046142
TCP	182556393	21,46515409
HTTP	126962291	14,92834677
SSL	18213037	2,14150619

LLC	9829155	1,155721381
TLSv1.2	1818378	0,213806612
MSMMS	876281	0,103033952
SSH	782181	0,091969585

Los protocolos más utilizados son UDP y TCP respectivamente. Esto es lógico ya que el uso de estos protocolos es necesario para otras conexiones HTTP o DNS. UDP proporciona una comunicación sin control de errores y, por defecto, en una red prevalece su uso sobre el de TCP ya que los datos en tiempo real utilizan la mayor parte del ancho de banda. Estos protocolos pertenecen al nivel de transporte.

Otro protocolo muy utilizado es HTTP. Este protocolo es utilizado para conexiones a páginas web y pertenece a la capa de aplicación.

El cuarto protocolo más utilizado es SSL perteneciente al nivel de transporte y que esta relacionado con el uso del protocolo HTTPS (http cifrado).



Resultados del análisis por IP.

Las graficas por IP como se ha comentado serían muy confusas y por ello solo se muestran las representaciones según el dominio de las webs. Para estas representaciones se ha incluido en un único archivo los resultados del número de paquetes según el dominio por minuto de todas las ejecuciones.

Por Redes Sociales:

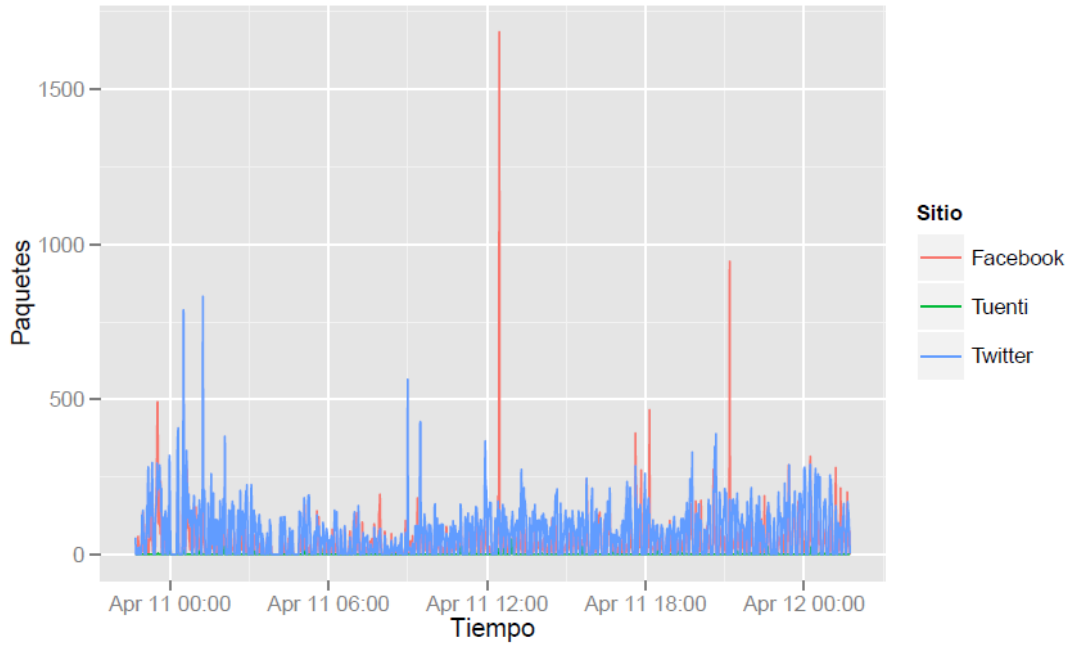


Figura 87: Trafico por redes sociales.

Como se percibe en la gráfica la red social más utilizada es Twitter a pesar de que Facebook es muy visitado en ciertos minutos. La red social Tuenti consigue escasas visitas en comparación con el resto.

Estas webs son muy utilizadas durante todo el día, a pesar de que parecen más utlizadas al principio de la noche.

Por webs de Noticias:

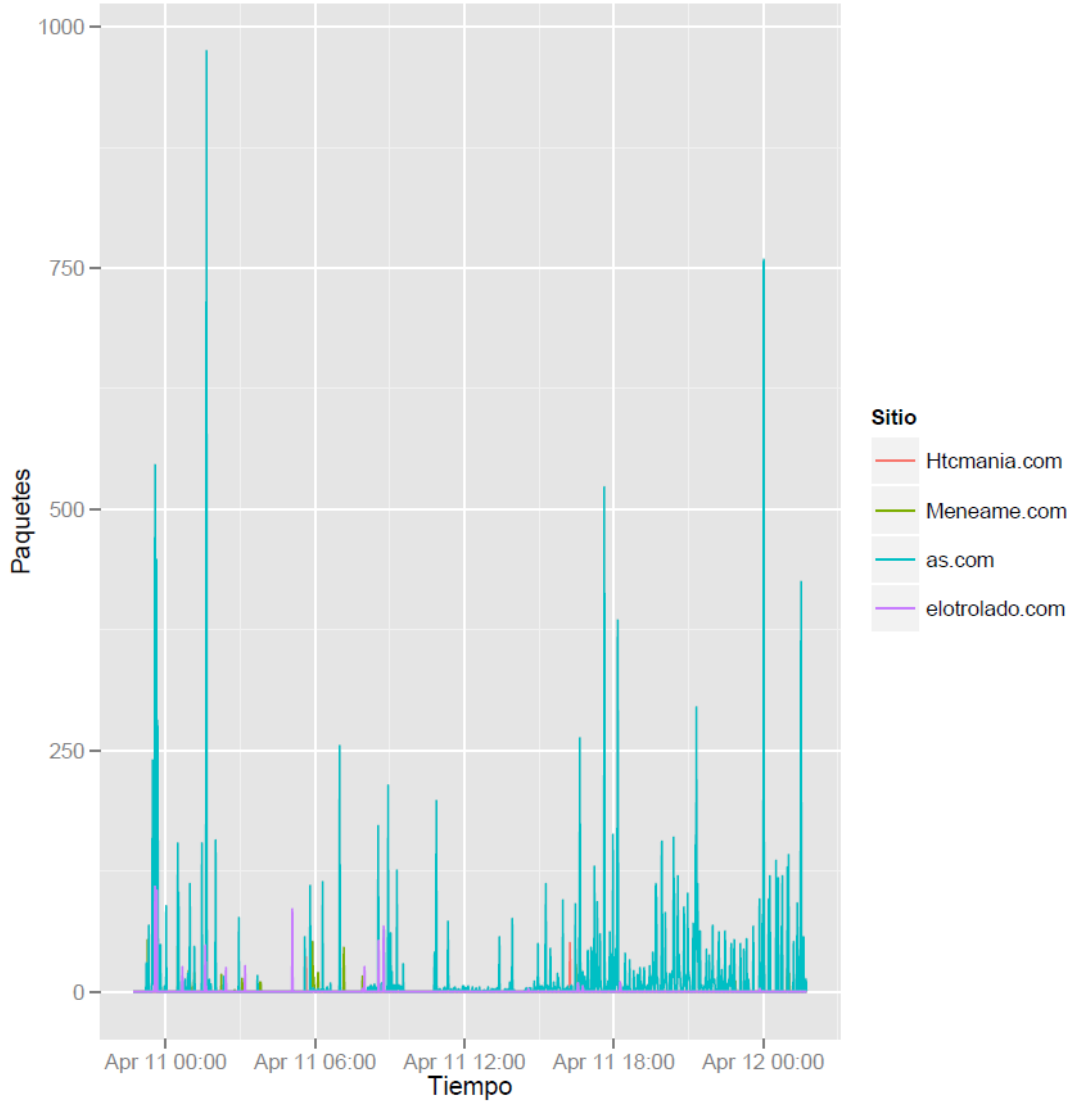


Figura 88: Trafico por webs noticias.

En la categoría de webs de noticas As.com destaca mucho sobre el resto. Las noticias deportivas son más apetecibles para el perfil de usuario del evento y es razonable que sea esta la web más visitada en la categoría.

Por webs de música:

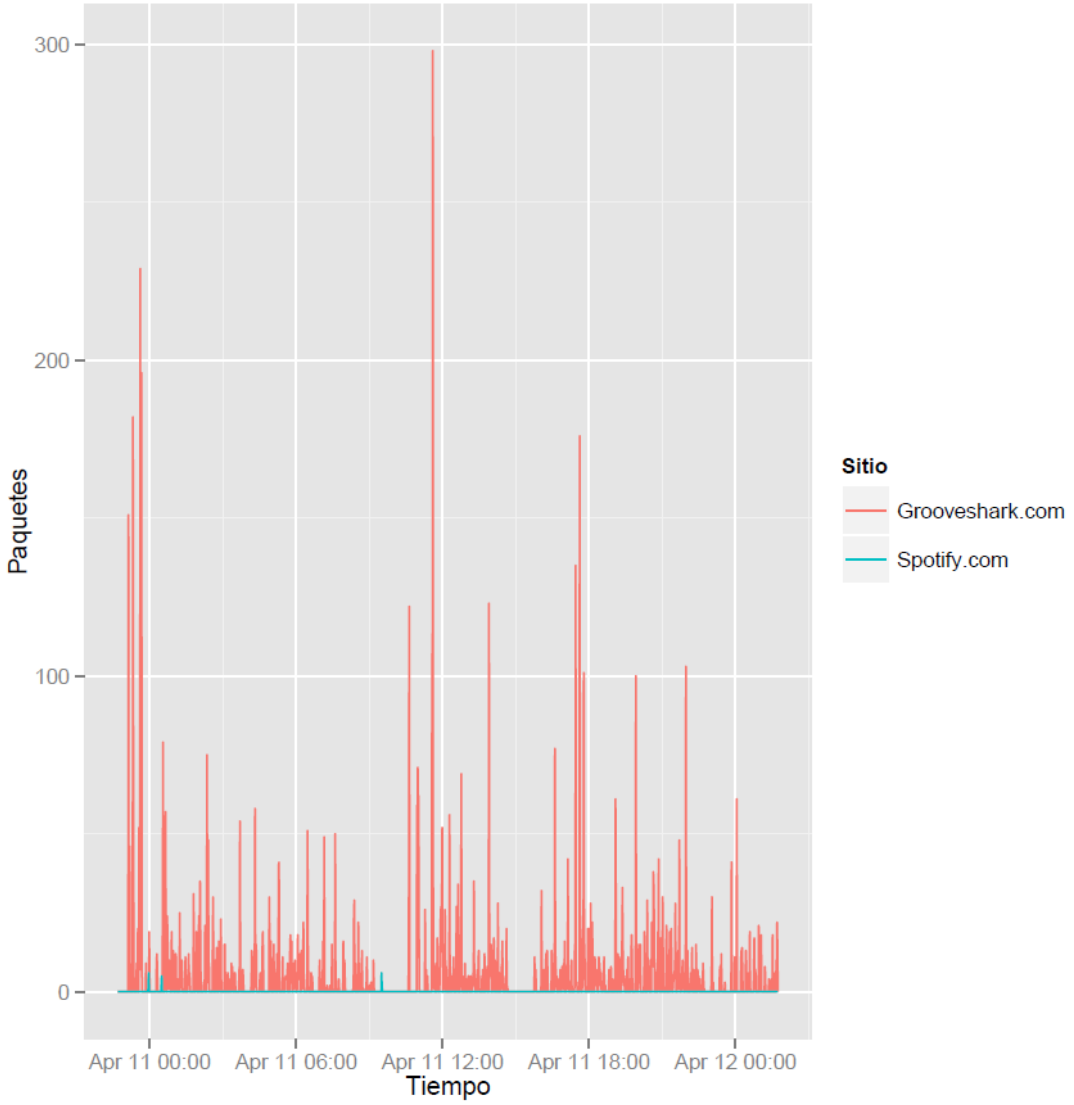


Figura 89: Trafico por webs de música.

Respecto a las webs de música Grooveshark.com es muy superior en número de visitas a Spotify.com. Un hecho que puede influir es que Grooveshark es de acceso gratuito al contrario que Spotify.

Por webs de cine:

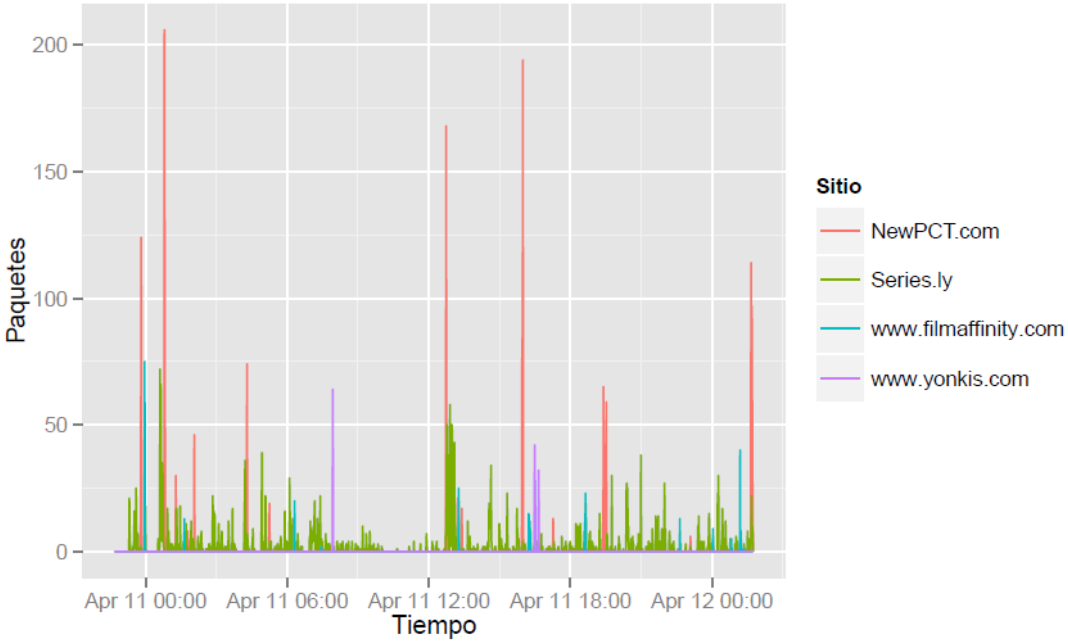


Figura 90: Tráfico por webs de cine.

En webs de cine Series.ly es la que en conjunto recibe más visitas a pesar de que NewPCT.com predomina en algunos momentos del día.

<u>Por webs de Juego:</u>

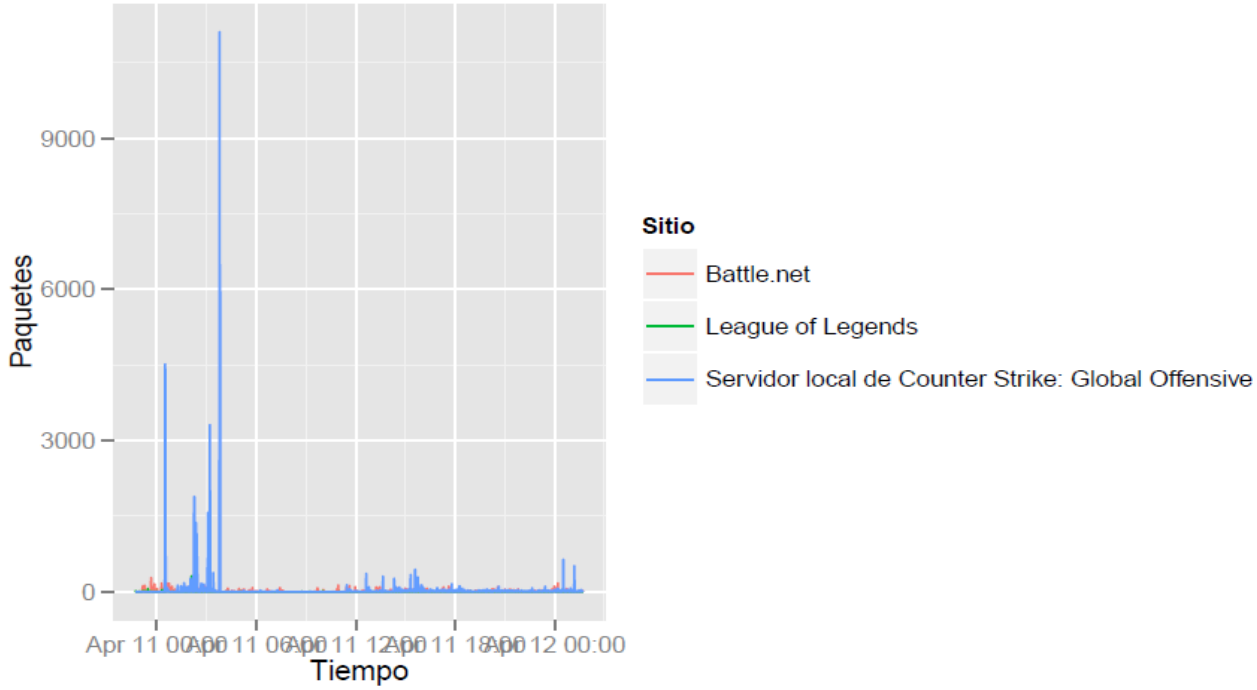


Figura 91: Tráfico por juegos.

En esta categoría Counter Strike destaca sobre los demás como el más utilizado.

Por correos electrónicos:

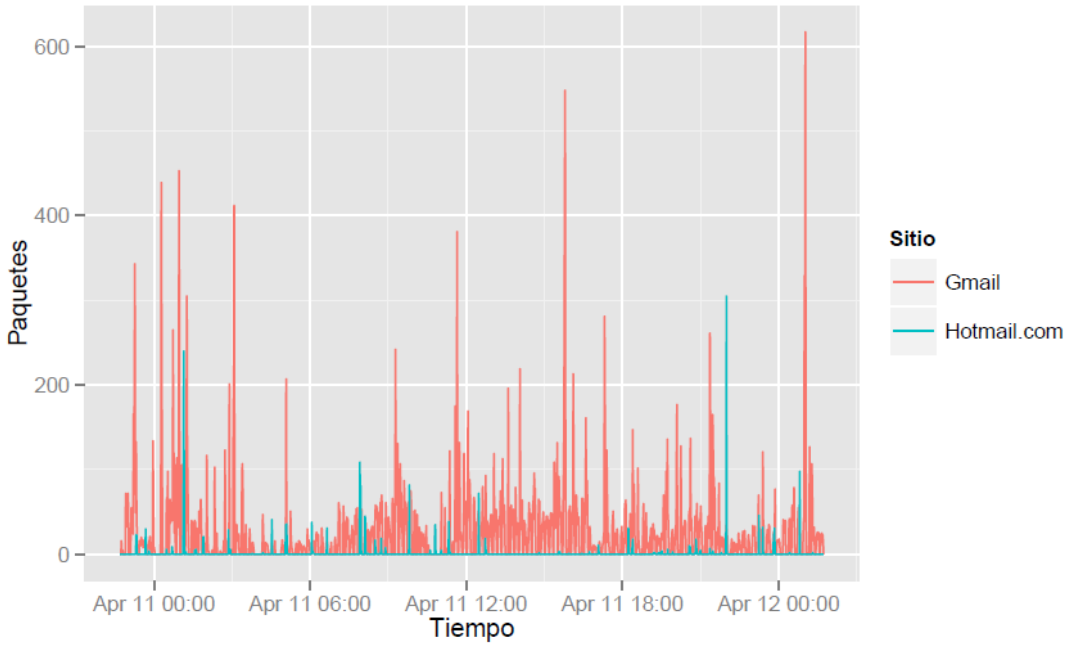


Figura 92: Tráfico por correos electrónicos.

En lo que a los correos electrónicos respecta, el correo de google Gmail es ampliamente el más utilizado. Hotmail no genera demasiadas visitas y su uso está descendiendo progresivamente.

Google y Youtube:

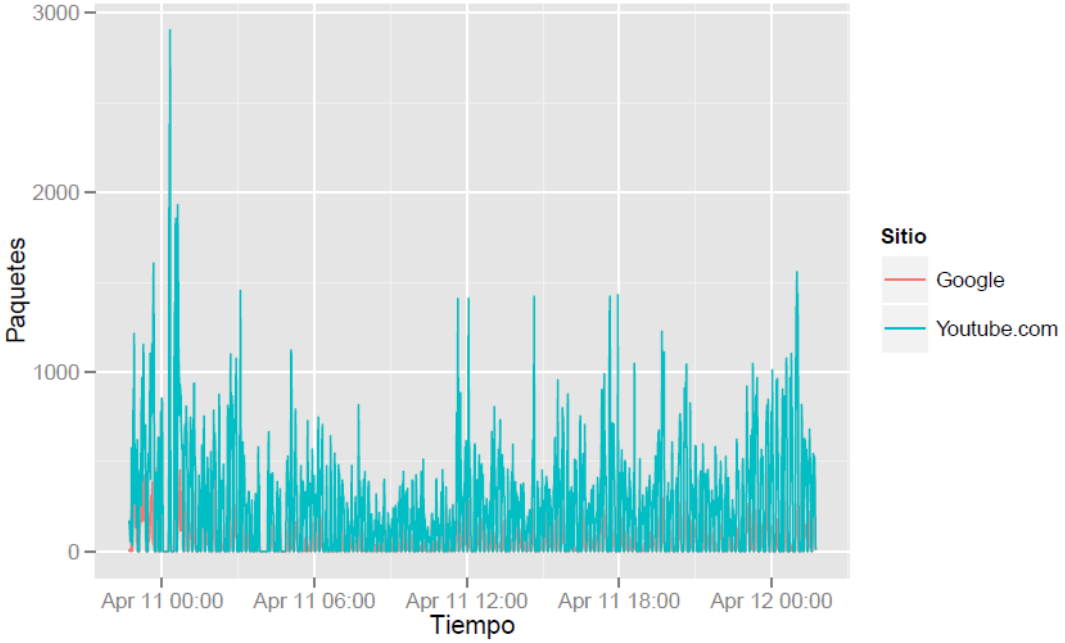


Figura 93: Tráfico web en Google y Youtube.

Si comparamos dos de las webs más conocidas mundialmente se ve como Youtube.com supera ampliamente en visitas a Google. Muchos usuarios no necesitan de Google más que para búsquedas muy concretas pues las paginas más frecuentemente utilizadas se suele acceder directamente a su dominio. Youtube, sin embargo, es utilizado para visualización de videos y en muchas ocasiones como reproductor de música.

Test de velocidad:

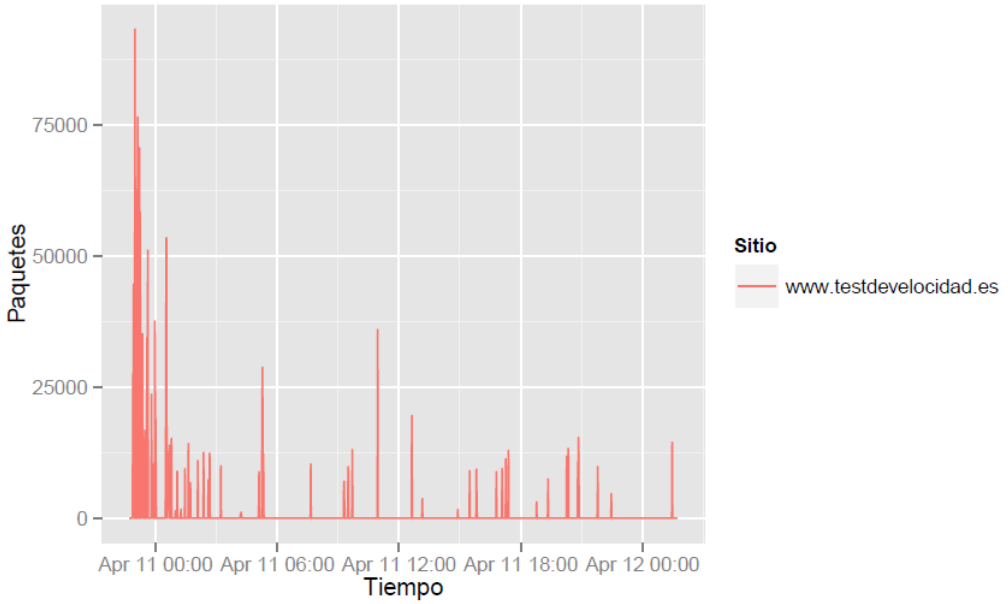


Figura 94: Tráfico para testdevelocidad.es

La web testdevelocidad.es es altamente utilizada al comienzo del evento puesto que todos los usuarios pretenden comprobar su velocidad de conexión. Durante todo este primer día la web es muy visitada.

Web de la Teleco LAN Party:

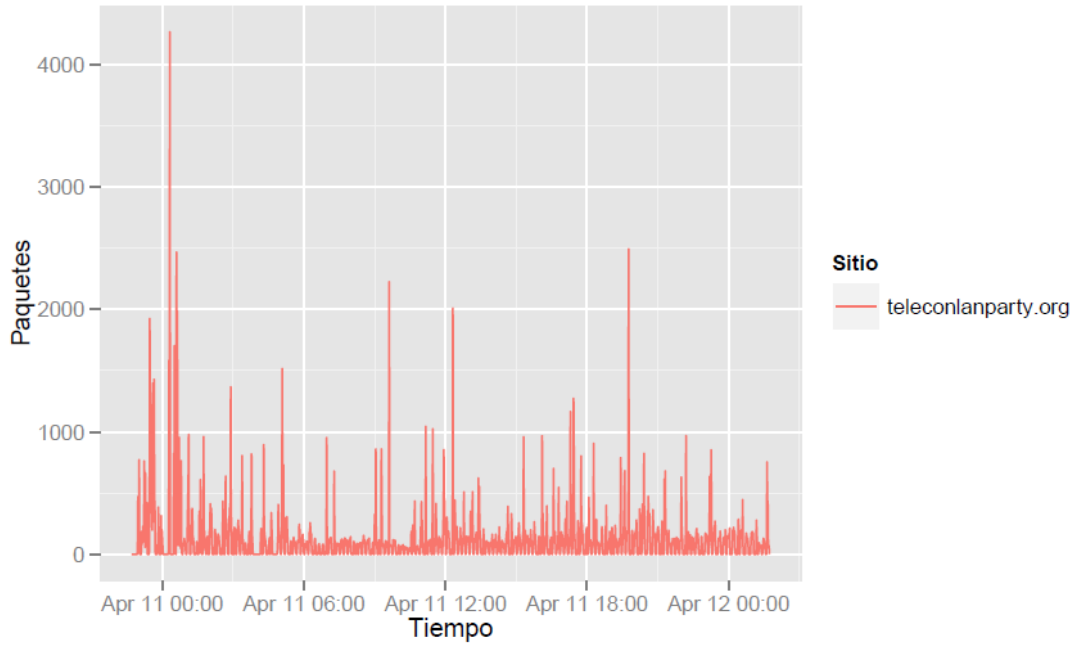


Figura 95: Tráfico web para teleconlanparty.org

La página web del evento es bastante visitada a lo largo de todo el día.

Las tres páginas más visitadas:

En la siguiente gráfica se muestra una grafica con las tres webs con mas tráfico.

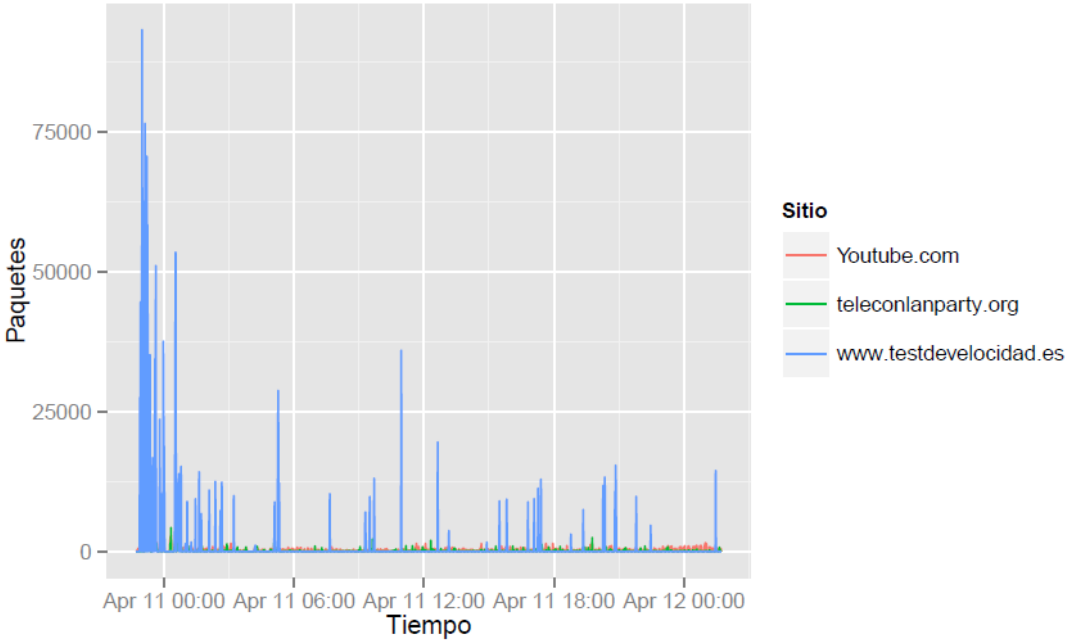
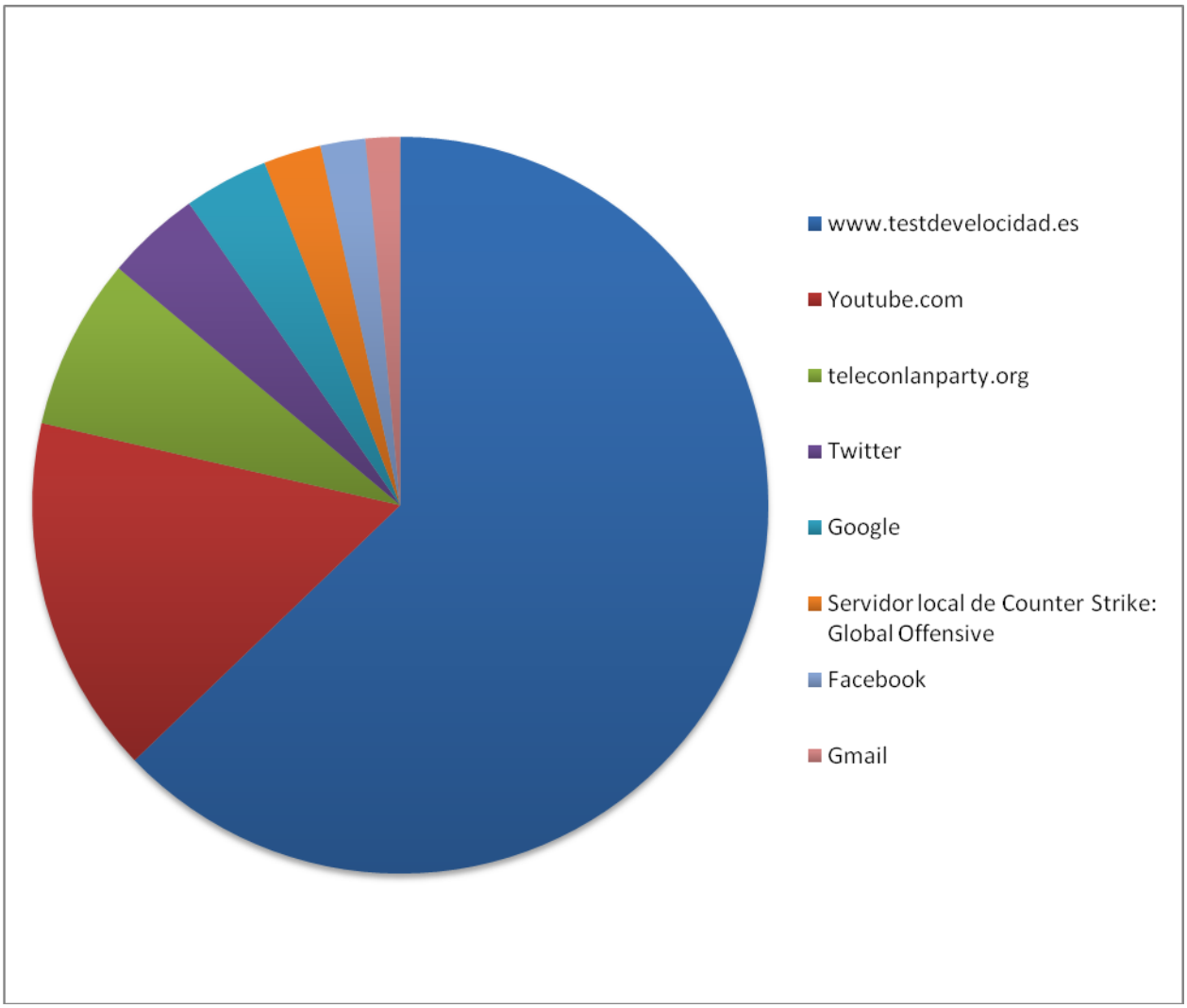


Figura 96: Tráfico en las tres webs más visitadas.

A continuación, se muestra una tabla que representa el porcentaje de tráfico respecto del total para las ocho webs mas utilizadas:

Dominio	Paquetes	Porcentaje (%)
www.testdevelocidad.es	1646645	62,8330018
Youtube.com	412866	15,75422154
teleconlanparty.org	197490	7,535862026
Twitter	108810	4,151993251
Google	97285	3,712220048
Servidor local de Counter Strike: Global Offensive	66122	2,523096202
Facebook	51702	1,972855023
Gmail	39749	1,516750112

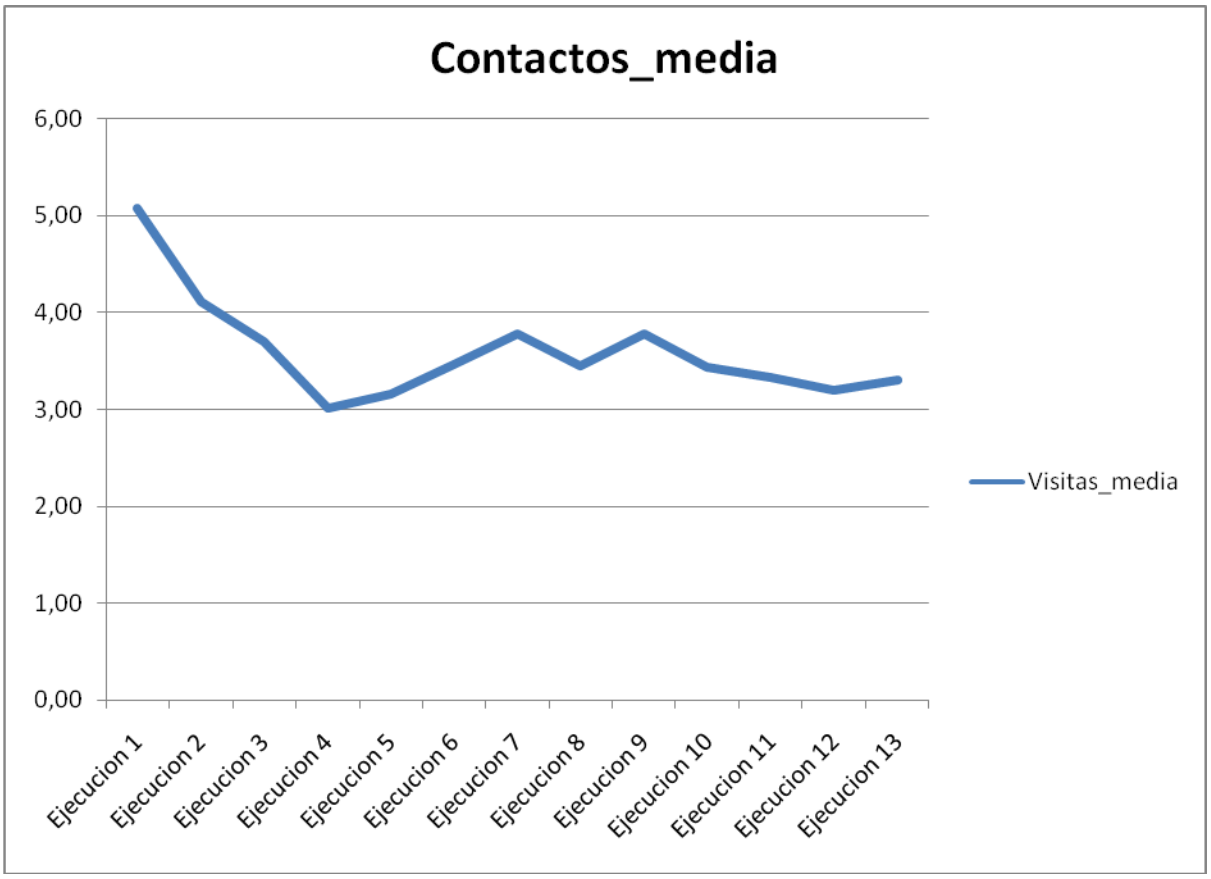


Como se puede comprobar la página testdevelocidad.es representa durante el primer día más del 60% del tráfico total para todos los dominios.

Finalmente se presenta el número medio de contactos con páginas distintas por usuario:

Numero Medio Contactos	Nº Ejecución
5,07	Ejecucion 1
4,11	Ejecucion 2
3,70	Ejecucion 3
3,01	Ejecucion 4
3,15	Ejecucion 5
3,46	Ejecucion 6

3,78	Ejecucion 7
3,45	Ejecucion 8
3,78	Ejecucion 9
3,43	Ejecucion 10
3,33	Ejecucion 11
3,19	Ejecucion 12
3,30	Ejecucion 13



Como se ha comentado anteriormente, este resultado tiene únicamente en cuenta los dominios de la lista utilizada. Por tanto, es probable que el número de contactos medio sea mas elevado provocado por los dominios que no entran en la lista auxiliar. Este resultado podría ser considerado una cota inferior del valor real, pese a ello, los dominios tenidos en cuenta son los más visitados por los usuarios de Internet.

Capítulo 5. Conclusiones y trabajos futuros.

En el presente trabajo fin de grado se ha presentado una de las herramientas más importantes dentro del entorno Big Data. El objetivo perseguido ha sido introducir una tecnología cuyo uso es muy extendido en la actualidad y que crecerá considerablemente en los próximos años. Se trata de una tecnología en pleno desarrollo donde están centrados los esfuerzos de numerosas personas y empresas a través de numerosas líneas de trabajo en relación con Hadoop. En este trabajo se ha intentado presentar las características más importantes así como la utilización del lenguaje de tratamiento de datos más novedoso dentro de la tecnología Big Data: Pig.

Debido a la experiencia adquirida con la realización del trabajo, se pueden destacar las principales bondades de las herramientas utilizadas y del trabajo realizado:

- Lenguaje muy dinámico: Tanto Pig como Hadoop en sí mismo, se adaptan realmente bien a cualquier tipo de datos. Los datos utilizados para evaluar la tecnología seguían una estructura muy concreta con numerosos obstáculos que se han conseguido solventar.

- Baja complejidad: A pesar de haber obtenido numerosos parámetros de los datos el Script desarrollado no contenía demasiadas líneas y la comprensión del código es relativamente sencilla.

- Se ha buscado un diseño muy sencillo y simple con el objetivo de poder mostrar al usuario las características de las dos plataformas sacrificando en algunas ocasiones la optimización de los procesos.

- Manejo de grandes cantidades de datos: La herramienta ha sido capaz de procesar bloques de 12 GB de manera eficiente a pesar de ejecutarse dentro de una máquina virtual.

- Alta escalabilidad: El Script creado podría ejecutarse en un clúster Hadoop sin problemas con el fin de poder procesar mayor cantidad de datos al mismo tiempo.

No obstante, el Script desarrollado requiere que los archivos de entrada sean preprocesados para obtener un esquema concreto. Este preprocesamiento podría haberse realizado con Pig y Hadoop, sin embargo, las limitaciones de disco duro y no disponer de un clúster de computadores obligaron a la búsqueda de una herramienta más eficiente en estos casos. El no disponer del hardware necesario para la instalación de un clúster ha supuesto la mayor limitación durante el desarrollo del trabajo debido al tamaño de los datos. Esto ha obligado a fragmentar los datos de partida para su análisis en una única computadora. La realización de un clúster de computadoras para el análisis de estos datos será propuesto como una línea de trabajo futura.

En cuanto a las conclusiones personales, previo al desarrollo del proyecto el estudio de los datos y, en concreto, las nuevas tecnologías de Big Data me han producido una especial atracción. Además de un gran interés en conceptos relacionados con la computación paralela: arquitecturas, paradigmas de programación o soluciones el aprovechamiento de los recursos en una red compartida. Este hecho me llevo al desarrollo del presente trabajo relacionado con la tecnología más novedosa en este panorama.

El trabajo realizado supone la tarea de mayores dimensiones a la cual me he enfrentado, esto me ha permitido adquirir habilidades en el procesamiento de datos así como aprender un nuevo lenguaje de programación. Además he adquirido conocimientos para el auto aprendizaje, la búsqueda de información y la investigación. Personalmente, me ha resultado muy gratificante ser capaz de solventar los problemas e inconvenientes que han surgido durante el desarrollo del trabajo. Por último, me gustaría señalar que ha resultado muy agradable poder ejercer la afición por el mundo de Big Data y realizar un trabajo fin de grado al mismo tiempo.

Para terminar, se van a definir algunas líneas de trabajo futuras relacionadas con el presente trabajo fin de grado:

- La primera propuesta va enfocada a solventar el problema de la ejecución de los datos. Se propone la instalación de un clúster de computadores con Hadoop que permitan analizar mayor cantidad de datos al mismo tiempo. Este trabajo puede suponer un punto de partida para la compresión tanto teórica como práctica de Hadoop y Pig facilitando de este modo la creación de un Script para otro tipo de datos.

- Otro posible trabajo sería la optimización del Script realizado mediante JAVA y comparar de este modo el tiempo de ejecución con esta tecnología. A pesar de que Pig es más apropiado para el análisis de datos, JAVA MapReduce permite mejorar el rendimiento de Hadoop notablemente.

- Como objetivo a largo plazo se podría crear una aplicación con una interfaz gráfica que permita realizar la instalación del clúster más automáticamente. Es decir, una aplicación que permita configurar el número de máquinas en el cluster, una clase map, una clase reduce y especificar los parámetros generales para poder lanzar un trabajo sobre un entorno distribuido de datos, aprovechando la capacidad de procesamiento que ofrezca éste.

- Un trabajo interesante sería la creación de un editor de texto para Pig que permita reconocer errores sintácticos y de estructura.

- Una aplicación muy actual sería la utilización de Hadoop para el análisis de información proveniente de redes sociales (web social media). Por ejemplo, se podría utilizar la Twitter API para extraer tweets y realizar un análisis de la

opinión de los usuarios sobre algún tema. Se podría aprovechar la incorporación del componente YARN a Hadoop para utilizar otro tipo de paradigma de programación. Plataformas como Storm o Giraph serían muy interesantes para este tipo de aplicaciones. Además se podría realizar un estudio de las herramientas mas utilizadas en Hadoop con la aparición de YARN.

Bibliografía.

[Ref1, WWW] http://www.r-project.org

[Ref2,WWW] http://www.cloudera.com/content/support/en/documentation/cdh4-documentation/cdh4-documentation-v4-latest.html

[Ref3,WWW] http://www.sas.com/offices/europe/spain/

[Ref4,WWW] http://www-01.ibm.com/software/es/analytics/spss/

[Ref5,WWW] http://www.mongodb.org/

[Ref6,WWW] http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-hadoop/

[Ref7,WWW] http://discoproject.org/

[Ref8] Hadoop: The definitive guide, Tom White. Ed. O´Reilly.

[Ref9,WWW] http://hortonworks.com/blog/introducing-apache-hadoop-yarn/

[Ref10,WWW] http://hortonworks.com/blog/apache-hadoop-yarn-concepts-and-applications/

[Ref11,WWW] http://hortonworks.com/products/hortonworks-sandbox/

[Ref12,WWW] https://www.virtualbox.org/wiki/Downloads

[Ref13,WWW] http://hortonworks.com/products/hortonworks-sandbox/#install

[Ref14,WWW] http://hadoop.apache.org/docs/r0.18.3/commands_manual.html#fs

[Ref15,WWW] http://hadoop.apache.org/docs/r0.18.3/commands_manual.html#fs

[Ref16] Programming Pig, Alan Gates. Ed. O´Reilley.

[Ref17,WWW] http://blog.mortardata.com/post/77915006384/introducing-the-pig-cheat-sheet

[Ref18,WWW] http://pig.apache.org/

[Ref19,WWW] https://www.gnu.org/software/sed/manual/sed.html

[Ref20,WWW] http://www.ubuntu.com/download/desktop.

[Ref21,WWW] http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[Ref22,WWW] http://www.oracle.com/technetwork/es/java/javase/downloads/jdk7-downloads-1880260.html

[Ref23,WWW] http://hadoop.apache.org/releases.html

[Ref24,WWW] http://hadoop.apache.org/docs/r0.18.3/commands_manual.html#fs

ANEXO A: Instalación y configuración de Hadoop en entorno Linux

En este Anexo, se va a instalar Hadoop en un SO Linux que a su vez estará instalado en una máquina virtual. Como software de virtualización se ha elegido de nuevo VirtualBox y para el SO ubuntu-12.04.4. Hadoop, como se ha comentado en la memoria, es una herramienta de código libre y fue creada en un principio unicamente para ser utilizada en un SO Linux. Actualmente, existe la posibilidad de instalar algunas versiones de Hadoop en otros SO como OS X. Sin embargo, aún sigue siendo Linux la mejor opción para utilizar Hadoop. En este apartado se va a realizar una instalación en modo pseudo-distribuido de Hadoop. Todo el proceso se realizará paso a paso para que cualquier usuario pueda instalar Hadoop sin problemas.

Para poder instalar Hadoop es recomendable cumplir con los siguientes requisitos técnicos:

- Memoria RAM: 4GB como mínimo aunque este apartado se ha realizado con un PC con 8GB (que es lo recomendable).
- Procesadores: 2 como mínimo es lo recomendable.
- Espacio disco duro: Es conveniente reservar unos 30GB de disco duro para Hadoop aunque depende del uso que se le dé podría ser necesario más.

A.1 Instalación.

Para comenzar, hay que descargar VirtualBox [Ref12, WWW]. Una vez instalado, se tendrá de nuevo la pantalla de la Figura 55.

A continuación, se va a proceder a instalar el SO en VirtualBox. Para ello hay que descargar el archivo de imagen de disco del SO [Ref20, WWW]. Una vez descargado el SO, se va a instalar en VirtualBox. Es necesario crear en VirtualBox una máquina virtual nueva. Se pulsa el botón nueva (ver Figura 97) y a continuación se seleccionan las características del SO que vamos a instalar.

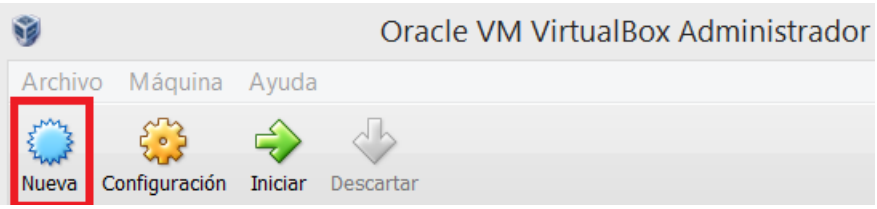


Figura 97: Barra de botones VirtualBox.

Se tendrá que seleccionar como tipo de SO Linux y la versión utilizada es Ubuntu (64 bit). Como nombre puede seleccionarse cualquiera, en este caso se llama a la máquina virtual hduser2 (Hadoop user 2). Se puede ver la ventana en la Figura 98.

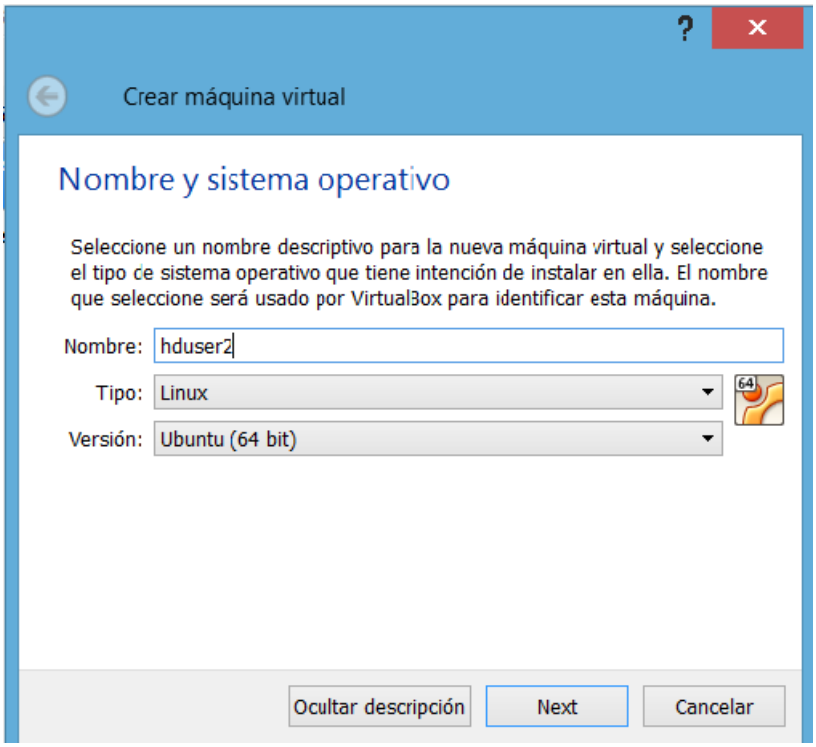


Figura 98: Nombre y SO VirtualBox.

Una vez completadas las características pedirá la cantidad de memoria RAM que queremos dedicar a la máquina virtual. En este caso se ha reservado 4096MB (ver Figura 99) de memoria ya que se dispone de un computador con 8GB pero sería suficiente una memoria RAM de 2GB si el ordenador tiene 4GB de memoria total. Se pulsa Next para continuar con la configuración.

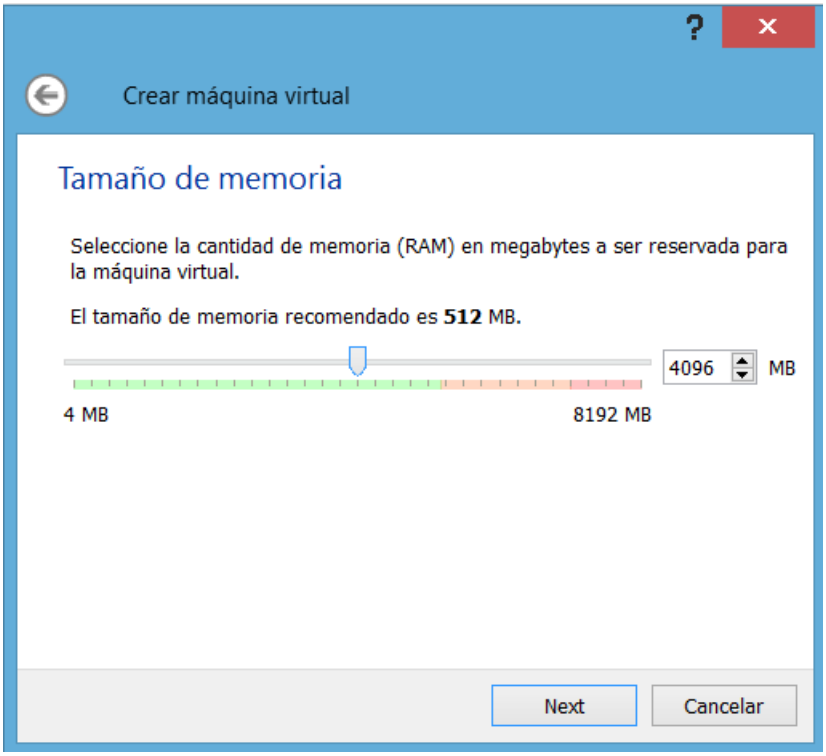


Figura 99: Memoria RAM VirtualBox.

El siguiente paso es crear una unidad de disco duro. Se selecciona crear un disco virtual ahora y se pulsa crear (ver Figura 100).

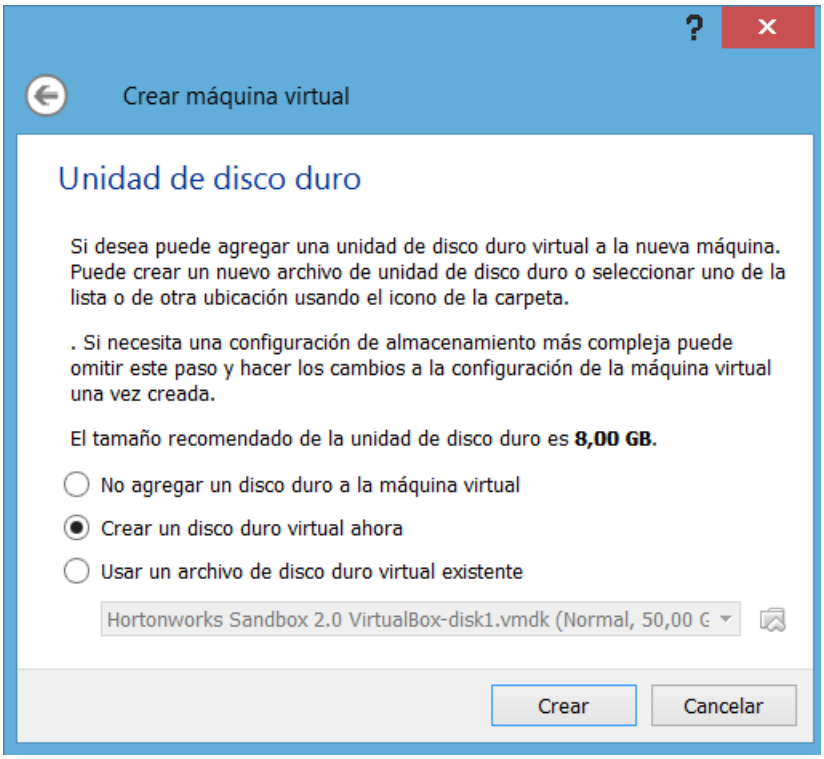


Figura 100: Unidad de Disco Duro VirtualBox.

Una vez creado el disco duro se pide el tipo de archivo. Se debe elegir VDI (VirtualBox disk image). En la siguiente opción elegir reservar dinámicamente el espacio del disco duro (ver Figura 101). Una vez seleccionadas estas dos opciones se debe elegir el tamaño del disco duro. Se elige en este caso 30GB.

Figura 101: Características Disco Duro VirtualBox.

Finalmente ya se tiene creada (ver Figura 102) la máquina virtual y únicamente se tiene que instalar Linux. Para ello, botón derecho sobre hduser2>configuración>Almacenamiento>Agregar dispositivo CD/DVD>Seleccionar Disco. Se busca el archivo de imagen previamente instalado y se selecciona Aceptar. Para iniciar el SO se selecciona Iniciar (ver Figura 103).

Seguidamente se tiene que seleccionar las características habituales para un SO: fecha, hora, contraseña, nombre de usuario, etc.

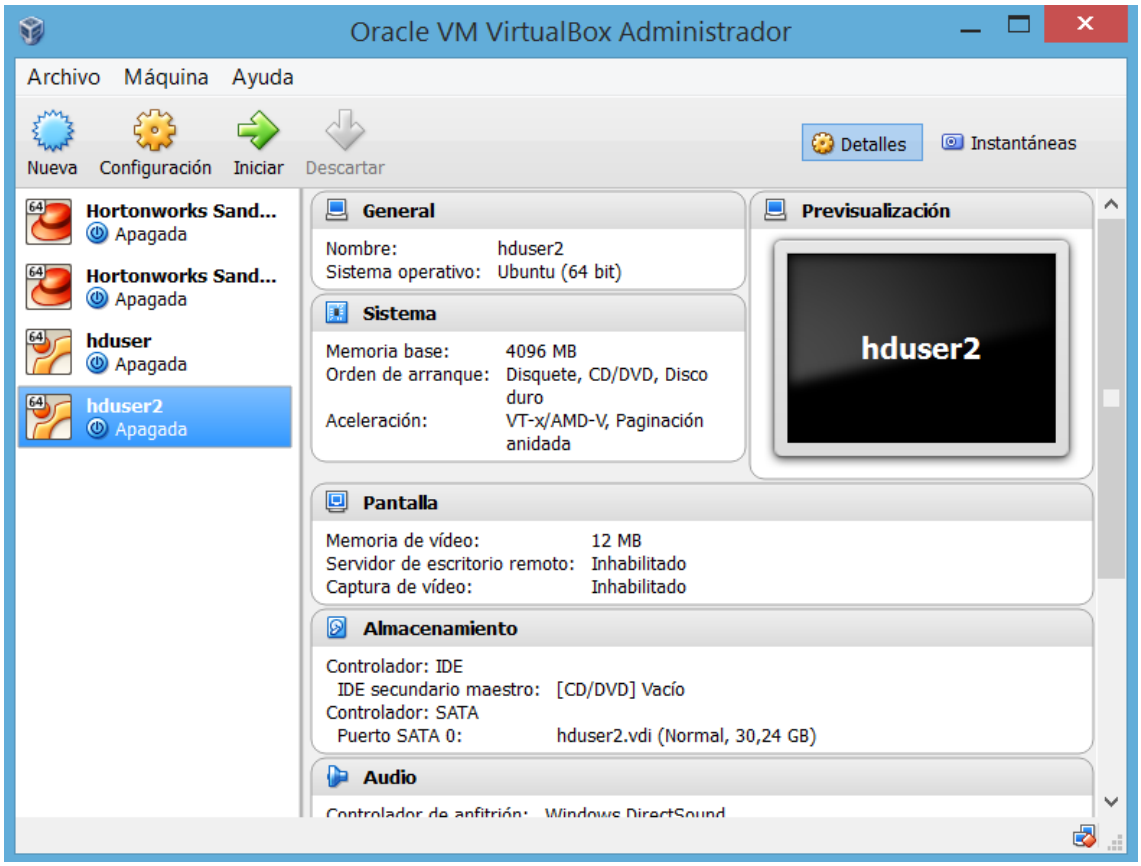


Figura 102: Máquina Virtual hduser2.

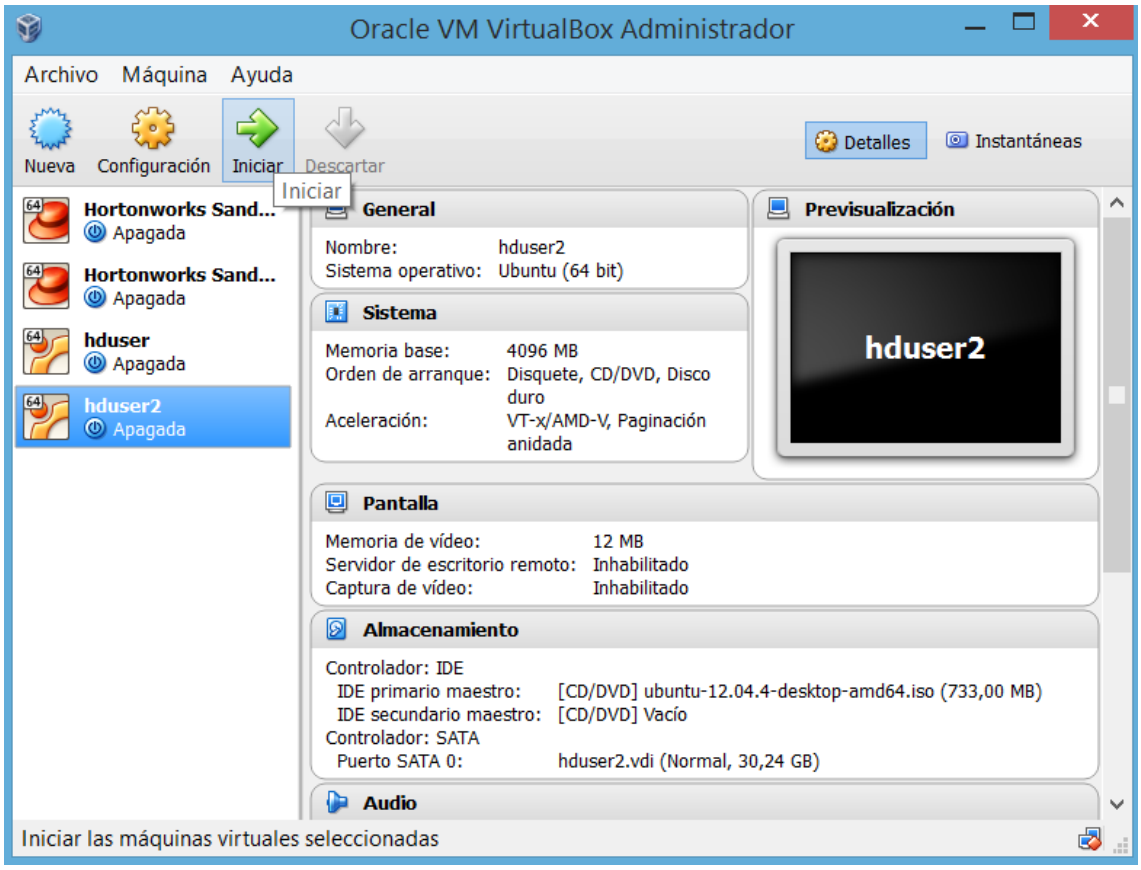


Figura 103: Iniciar Máquina Virtual VirtualBox.

Una vez configurado el SO se muestra el escritorio de Linux (Figura 104).

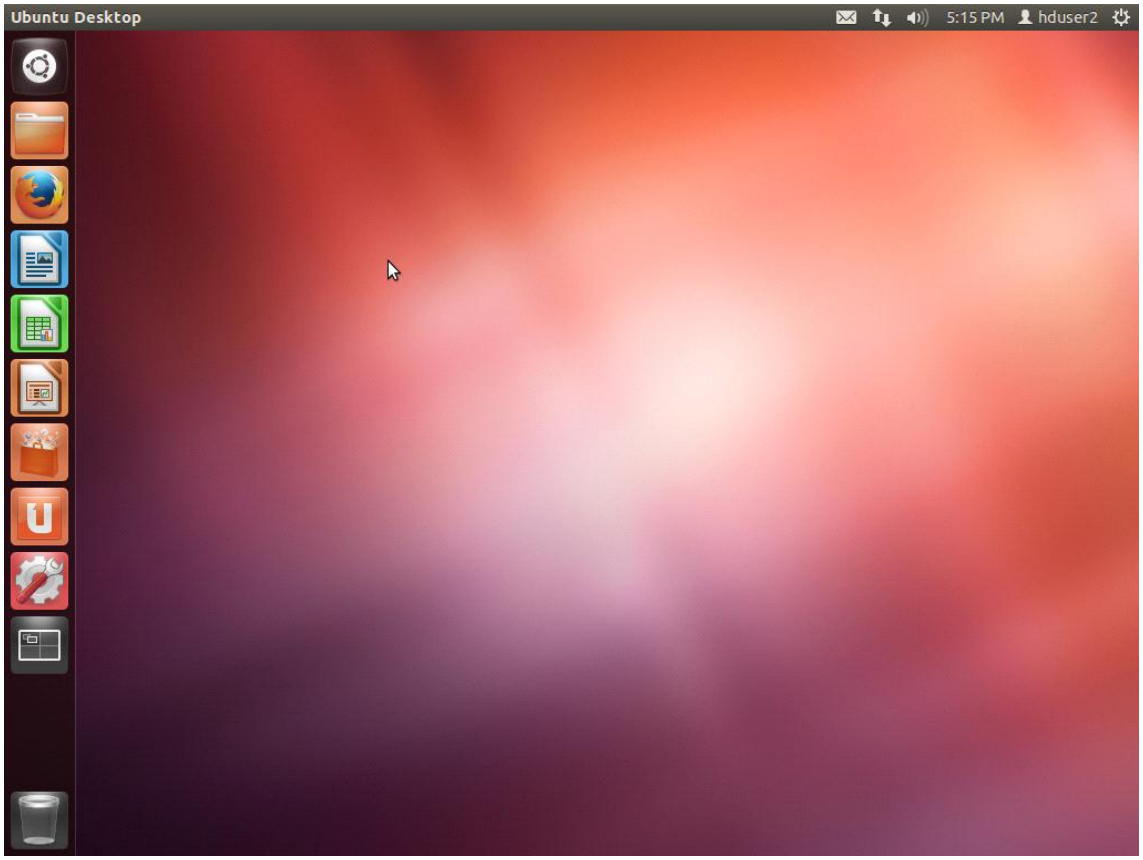


Figura 104: Escritorio Ubuntu.

A continuación, se puede instalar Hadoop en la máquina virtual [Ref21, WWW].

Hadoop necesita Java para ejecutarse. Para la realización del trabajo se ha utilizado la versión 1.7. En el sitio oficial de Java se puede descargar la máquina virtual [Ref22, WWW]. Se debe de comprobar que el formato del archivo descargado es .tar.gz, el formato de archivos comprimidos en Linux. Para instalar JAVA se inicia un terminal Dash Home> Terminal (ver Figura 105).

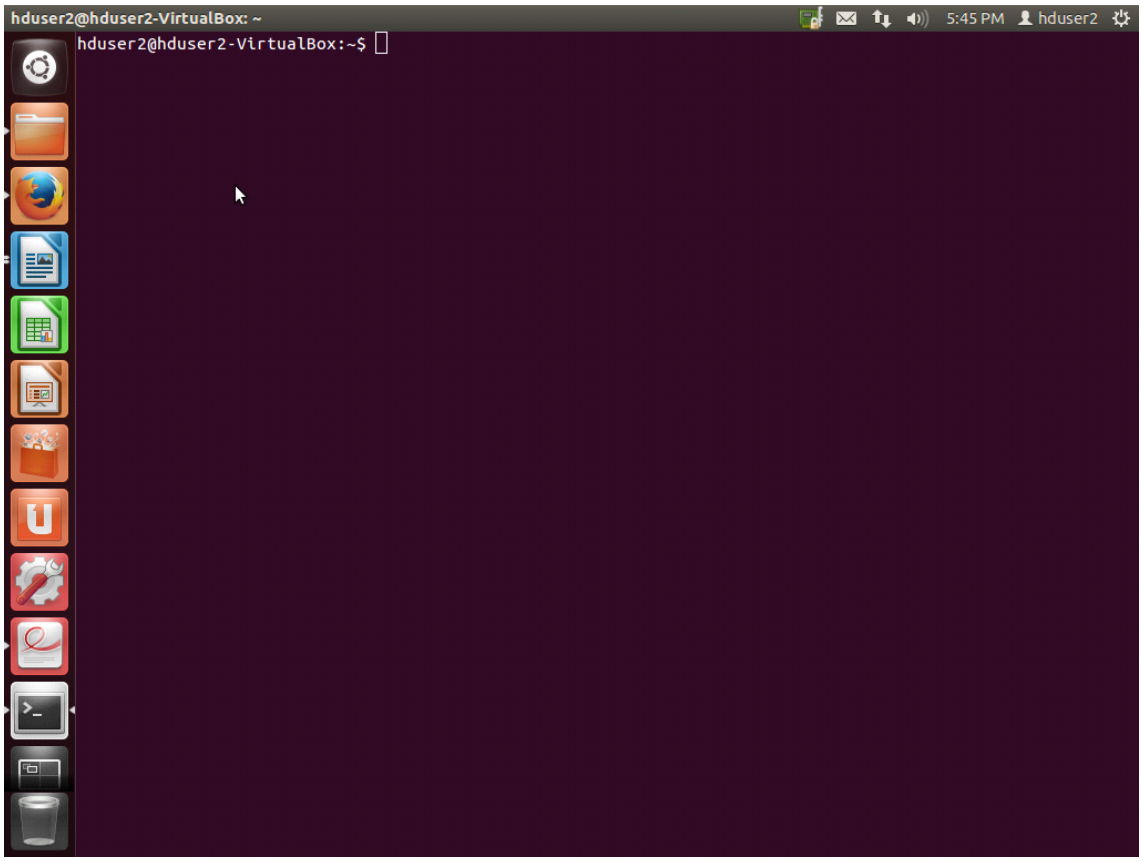


Figura 105: Terminal Ubuntu.

A continuación, hay que situarse en el directorio donde se ha descargado el JDK, en este caso Downloads:

```
hduser2@hduser2-VirtualBox:~$ cd Downloads
hduser2@hduser2-VirtualBox:~/Downloads$
```

Se descomprime el archivo descargado:

```
hduser2@hduser2-VirtualBox:~/Downloads$ tar -xvf jdk-7u21-linux-
x64.tar.gz
```

Se mueve el JDK al directorio usr/lib (Necesario escribir la contraseña de usuario cuando la solicita):

```
hduser2@hduser2-VirtualBox:~/Downloads$ sudo mkdir -p /usr/lib/jvm
[sudo] password for hduser2:
hduser2@hduser2-VirtualBox:~/Downloads$ sudo mv ./jdk1.7.0_21
/usr/lib/jvm/jdk1.7.0
```

Se instalan las posibles actualizaciones:

```
hduser2@hduser2-VirtualBox:~/Downloads$ sudo update-alternatives --
install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.7.0/bin/java" 1
update-alternatives: using /usr/lib/jvm/jdk1.7.0/bin/java to provide
/usr/bin/java (java) in auto mode.
hduser2@hduser2-VirtualBox:~/Downloads$ sudo update-alternatives --
install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk1.7.0/bin/javac" 1
update-alternatives: using /usr/lib/jvm/jdk1.7.0/bin/javac to provide
/usr/bin/javac (javac) in auto mode.
hduser2@hduser2-VirtualBox:~/Downloads$ sudo update-alternatives --
install "/usr/bin/javaws" "javaws" "/usr/lib/jvm/jdk1.7.0/bin/javaws" 1
update-alternatives: using /usr/lib/jvm/jdk1.7.0/bin/javaws to provide
/usr/bin/javaws (javaws) in auto mode.
```

Se ejecutan los siguientes comandos para corregir los permisos de los ejecutables:

```
hduser2@hduser2-VirtualBox:~/Downloads$ sudo chmod a+x /usr/bin/java
hduser2@hduser2-VirtualBox:~/Downloads$ sudo chmod a+x /usr/bin/javac
hduser2@hduser2-VirtualBox:~/Downloads$ sudo chmod a+x
/usr/bin/javaws
hduser2@hduser2-VirtualBox:~/Downloads$ sudo chown -R root:root
/usr/lib/jvm/jdk1.7.0
```

Se comprueba la versión de JAVA:

```
hduser2@hduser2-VirtualBox:~/Downloads$ java -version
java version "1.7.0_21"
Java(TM) SE Runtime Environment (build 1.7.0_21-b11)
Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed
mode)
```

Como se ve se ha instalado la versión 1.7.0_21 de java.

A continuación es necesario instalar y configurar el protocolo ssh. Hadoop requiere que los procesos puedan comunicarse de forma segura entre sí con los diferentes equipos en la red, y para ello es necesaria la utilización de ssh. Por tanto, es necesario asegurarse que se puede iniciar sesión ssh sin contraseña en las máquinas del clúster. A

continuación se procede a realizar la configuración para permitir la comunicación ssh entre las distintas máquinas.

Se instala el cliente ssh:

```
hduser2@hduser2-VirtualBox:~/Downloads$ sudo apt-get install openssh-
client
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  libpam-ssh keychain monkeysphere openssh-blacklist openssh-blacklist-
extra
The following packages will be upgraded:
  openssh-client
1 upgraded, 0 newly installed, 0 to remove and 133 not upgraded.
Need to get 942 kB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://es.archive.ubuntu.com/ubuntu/ precise-updates/main
openssh-client amd64 1:5.9p1-5ubuntu1.4 [942 kB]
Fetched 942 kB in 1s (492 kB/s)
(Reading database ... 144272 files and directories currently
installed.)
Preparing to replace openssh-client 1:5.9p1-5ubuntu1.1 (using
.../openssh-client_1%3a5.9p1-5ubuntu1.4_amd64.deb) ...
Unpacking replacement openssh-client ...
Processing triggers for man-db ...
Setting up openssh-client (1:5.9p1-5ubuntu1.4) ...
```

Se instala el servidor ssh:

```
hduser2@hduser2-VirtualBox:~/Downloads$ sudo apt-get install
openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  ssh-import-id
Suggested packages:
  rssh molly-guard openssh-blacklist openssh-blacklist-extra
monkeysphere
The following NEW packages will be installed:
  openssh-server ssh-import-id
0 upgraded, 2 newly installed, 0 to remove and 133 not upgraded.
Need to get 346 kB of archives.
After this operation, 881 kB of additional disk space will be
used.
Do you want to continue [Y/n]? Y
```

```
Get:1 http://es.archive.ubuntu.com/ubuntu/ precise-updates/main
openssh-server amd64 1:5.9p1-5ubuntu1.4 [339 kB]
Get:2 http://es.archive.ubuntu.com/ubuntu/ precise/main ssh-import-id
all 2.10-0ubuntu1 [6,598 B]
Fetched 346 kB in 0s (584 kB/s)
Preconfiguring packages ...
Selecting previously unselected package openssh-server.
(Reading database ... 144272 files and directories currently
installed.)
Unpacking openssh-server (from .../openssh-server_1%3a5.9p1-
5ubuntu1.4_amd64.deb) ...
Selecting previously unselected package ssh-import-id.
Unpacking ssh-import-id (from .../ssh-import-id_2.10-0ubuntu1_all.deb)
...
Processing triggers for ureadahead ...
Processing triggers for ufw ...
Processing triggers for man-db ...
Setting up openssh-server (1:5.9p1-5ubuntu1.4) ...
Creating SSH2 RSA key; this may take some time ...
Creating SSH2 DSA key; this may take some time ...
Creating SSH2 ECDSA key; this may take some time ...
ssh start/running, process 4251
Setting up ssh-import-id (2.10-0ubuntu1) ...
```

Para la configuración del ssh, primero se genera una clave ssh con un password vacío:

```
hduser2@hduser2-VirtualBox:~/Downloads$ su - hduser2
Password:
hduser2@hduser2-VirtualBox:~$ ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser2/.ssh/id_rsa):
/home/hduser2/.ssh/id_rsa
Created directory '/home/hduser2/.ssh'.
Your identification has been saved in /home/hduser2/.ssh/id_rsa.
Your public key has been saved in /home/hduser2/.ssh/id_rsa.pub.
The key fingerprint is:
81:0f:a1:ae:71:de:b2:f8:3d:23:4b:2f:62:88:b5:8b hduser2@hduser2-
VirtualBox
The key's randomart image is:
+--[ RSA 2048]----+
|      .          |
|     . o         |
|    . o .        |
|   .   o .       |
|  . o   S        |
|  .= .           |
|.o..+ .          |
|o.+oo=o          |
|E.o+++oo         |
+-----------------+
```

Una vez generada la clave pública es necesario guardarla en el archivo de claves autorizadas para el usuario actual en cada una de las máquinas que forman parte del cluster:

```
hduser2@hduser2-VirtualBox:~$ cat $HOME/.ssh/id_rsa.pub >>
$HOME/.ssh/authorized_keys
```

Se prueba la conexión a localhost para comprobar que se ha configurado todo correctamente. No nos debería pedir contraseña:

```
hduser2@hduser2-VirtualBox:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is ea:45:6b:c7:51:53:2c:41:0b:ab:5c:e0:6a:ef:a6:4d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known
hosts.
Welcome to Ubuntu 12.04.4 LTS (GNU/Linux 3.11.0-15-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

137 packages can be updated.
72 updates are security updates.
```

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

A continuación, se va a deshabilitar el protocolo IPv6 ya que es una práctica recomendable cuando se trabaja con Hadoop. Apache Hadoop trabaja con IPv4 y pueden ocurrir problemas si no se deshabilita IPv6. Para ello se ejecutan los comandos:

```
hduser2@hduser2-VirtualBox:~$ echo "net.ipv6.conf.all.disable_ipv6 =
1" | sudo tee -a /etc/sysctl.conf
net.ipv6.conf.all.disable_ipv6 = 1
hduser2@hduser2-VirtualBox:~$ echo
"net.ipv6.conf.default.disable_ipv6 = 1" | sudo tee -a
/etc/sysctl.conf
net.ipv6.conf.default.disable_ipv6 = 1
hduser2@hduser2-VirtualBox:~$ echo "net.ipv6.conf.lo.disable_ipv6 =
1" | sudo tee -a /etc/sysctl.conf
net.ipv6.conf.lo.disable_ipv6 = 1
```

Para que los cambios surjan efecto se debe reiniciar Ubuntu. Para comprobar que se ha deshabilitado IPv6 ejecutamos el comando:

```
hduser2@hduser2-VirtualBox:~$ cat
/proc/sys/net/ipv6/conf/all/disable_ipv6
1
```

Si devuelve un valor de 1 indica que IPv6 se ha deshabilitado, por contra, si devuelve 0 indica que IPv6 está activado.

Hasta ahora, se ha instalado Java, se ha configurado e instalado ssh y deshabilitado IPv6. A continuación ya es posible comentar a instalar y configurar Hadoop. Se comienza descargando la versión que se prefiera de Hadoop desde la página oficial [Ref23,WWW].

Hasta este punto, la instalación de cualquier versión de Hadoop sigue los mismos pasos. En primer lugar se va a explicar la configuración e instalación de Hadoop 1.2.1 y, posteriormente, se comentarán las diferencias para poder instalar cualquier versión de Hadoop a partir de la versión 2.0.

Para comenzar es necesario descomprimir el archivo que se ha descargado:

```
hduser2@hduser2-VirtualBox:~/Downloads$ sudo tar xzf hadoop-1.2.1.tar.gz
```

Se mueve todo al directorio /home:

```
hduser2@hduser2-VirtualBox:~/Downloads$ cd /usr/local
hduser2@hduser2-VirtualBox:/usr/local$ sudo mv
/home/hduser2/Downloads/hadoop-1.2.1 hadoop
```

Se crea el grupo hadoop:

```
hduser2@hduser2-VirtualBox:/usr/local$ sudo addgroup hadoop
Adding group `hadoop' (GID 1001) ...
Done.
hduser2@hduser2-VirtualBox:/usr/local$ sudo chown -R hduser2:hadoop
hadoop
```

Seguidamente, se va a configurar los archivos necesarios para un correcto

funcionamiento de Hadoop.

Se comienza configurando el archivo .bashrc que permite ejecutar comandos cada vez que realizamos login. Se abre el archivo con el comando:

```
hduser2@hduser2-VirtualBox:~$ gksudo gedit .bashrc
```

Se añade lo siguiente al final del documento:

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
export PIG_HOME=/usr/local/pig
export PIG_CLASSPATH=/usr/local/hadoop/conf

# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop
later on)
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0/

# Some convenient aliases and functions for running Hadoop-related
commands
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}

# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$PIG_HOME/bin
```

El próximo archivo que se va a modificar es hadoop-env.sh. Este script permite cargar las variables de entorno necesarias para la configuración. Se abre el archivo:

```
hduser2@hduser2-VirtualBox:~$ gksudo gedit /usr/local/hadoop/conf/hadoop-
env.sh
```

Se añade al final del documento lo siguiente:

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0/
```

El siguiente archivo es core-site.xml. Este fichero determina la url del namenode, además del puerto por el que se escuchan las peticiones de los clientes. También se configura cual es el directorio base para datos temporales. Se abre el archivo:

```
hduser2@hduser2-VirtualBox:~$ gksudo gedit /usr/local/hadoop/conf/core-
site.xml
```

Y se añade entre la marca de configuración (<configuration>) el siguiente (ver Figura 106):

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uri's scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uri's authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>
```

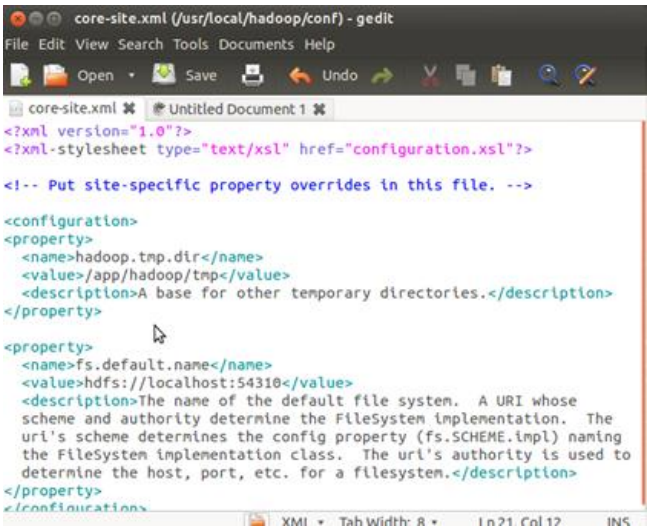


Figura 106: Fichero core-site.xml

El penúltimo archivo que se debe modificar es mapred-site.xml. Este fichero permite configurar la ubicación del jobtracker, entre otras. Se abre el archivo:

```
hduser2@hduser2-VirtualBox:~$ gksudo gedit
/usr/local/hadoop/conf/mapred-site.xml
```

Y se añade entre la marca <configuration></configuration> lo siguiente:

```
<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
</property>
```

El último fichero es hdfs-site.xml. En este fichero almacena, entre otros datos interesantes, la localización de los bloques de datos del HDFS. Se abre el archivo:

```
hduser2@hduser2-VirtualBox:~$ gksudo gedit /usr/local/hadoop/conf/hdfs-
site.xml
```

Y se añade lo siguiente entre la marca <configuration></configuration>:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is
created.
  The default is used if replication is not specified in create time.
  </description>
</property>
```

Una vez se ha configurado Hadoop, se tiene que crear una nueva carpeta que se ha especificado en el fichero core-site.xml. Para crear una carpeta con los correspondientes permisos se ejecutan los siguientes comandos:

```
hduser2@hduser2-VirtualBox:~$ sudo mkdir -p /app/hadoop/tmp
hduser2@hduser2-VirtualBox:~$ sudo chown hduser2:hadoop /app/hadoop/tmp
hduser2@hduser2-VirtualBox:~$ sudo chmod 750 /app/hadoop/tmp
```

Ahora para poder utilizar Hadoop, se debe formatear el HDFS:

```
hduser2@hduser2-VirtualBox:~$ /usr/local/hadoop/bin/hadoop namenode -
format
14/06/16 19:44:03 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = hduser2-VirtualBox/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.2.1
STARTUP_MSG:   build =
https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.2 -r
1503152; compiled by 'mattf' on Mon Jul 22 15:23:09 PDT 2013
STARTUP_MSG:   java = 1.7.0_21
************************************************************/
14/06/16 19:44:03 INFO util.GSet: Computing capacity for map BlocksMap
14/06/16 19:44:03 INFO util.GSet: VM type       = 64-bit
14/06/16 19:44:03 INFO util.GSet: 2.0% max memory = 1013645312
14/06/16 19:44:03 INFO util.GSet: capacity      = 2^21 = 2097152 entries
14/06/16 19:44:03 INFO util.GSet: recommended=2097152, actual=2097152
14/06/16 19:44:04 INFO namenode.FSNamesystem: fsOwner=hduser2
14/06/16 19:44:04 INFO namenode.FSNamesystem: supergroup=supergroup
14/06/16 19:44:04 INFO namenode.FSNamesystem: isPermissionEnabled=true
14/06/16 19:44:04 INFO namenode.FSNamesystem:
dfs.block.invalidate.limit=100
14/06/16 19:44:04 INFO namenode.FSNamesystem: isAccessTokenEnabled=false
accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
14/06/16 19:44:04 INFO namenode.FSEditLog:
dfs.namenode.edits.toleration.length = 0
14/06/16 19:44:04 INFO namenode.NameNode: Caching file names occuring
more than 10 times
14/06/16 19:44:04 INFO common.Storage: Image file
/app/hadoop/tmp/dfs/name/current/fsimage of size 113 bytes saved in 0
seconds.
14/06/16 19:44:04 INFO namenode.FSEditLog: closing edit log: position=4,
editlog=/app/hadoop/tmp/dfs/name/current/edits
14/06/16 19:44:04 INFO namenode.FSEditLog: close success: truncate to 4,
editlog=/app/hadoop/tmp/dfs/name/current/edits
14/06/16 19:44:04 INFO common.Storage: Storage directory
/app/hadoop/tmp/dfs/name has been successfully formatted.
14/06/16 19:44:04 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hduser2-VirtualBox/127.0.1.1
************************************************************/
```

Hadoop ya estaría instalado y podría ser utilizado. Para inicializar todos los componentes Hadoop se utiliza el comando:

```
hduser2@hduser2-VirtualBox:~$ /usr/local/hadoop/bin/start-all.sh
starting namenode, logging to /usr/local/hadoop/libexec/../logs/hadoop-
hduser2-namenode-hduser2-VirtualBox.out
localhost: starting datanode, logging to
/usr/local/hadoop/libexec/../logs/hadoop-hduser2-datanode-hduser2-
VirtualBox.out
localhost: starting secondarynamenode, logging to
/usr/local/hadoop/libexec/../logs/hadoop-hduser2-secondarynamenode-
hduser2-VirtualBox.out
starting jobtracker, logging to /usr/local/hadoop/libexec/../logs/hadoop-
hduser2-jobtracker-hduser2-VirtualBox.out
localhost: starting tasktracker, logging to
/usr/local/hadoop/libexec/../logs/hadoop-hduser2-tasktracker-hduser2-
VirtualBox.out
```

En este punto, se tendría Hadoop listo para empezar a trabajar. Sería importante comentar que Hadoop dispone de algunas interfaces web en las que se puede consultar de forma gráfica información relativa al NameNode, JobTracker y TaskTracker. Para poder acceder se pueden escribir las siguientes direcciones [Ref15, WWW]:

- NameNode: http://localhost:50070/
- JobTracker: http://localhost:50030/
- TaskTracker: http://localhost:50060/

Tras haber instalado la versión 1.2.1, se va a explicar los distintos pasos que se tendrían que seguir en caso de instalar una versión a partir de la 2.0. La única diferencia radica en los archivos a configurar. La instalación de Java, ssh, la deshabilitación de IPv6 y la modificación del fichero .bashrc seguirían los mismos pasos. En esta instalación el nombre de usuario es hduser en lugar de hduser2. A continuación se explica la manera de proceder para los demás archivos:

- hadoop-env.sh:

Para abrir el archivo:

```
hduser@hduser-VirtualBox:~$ gksudo gedit
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

Se añadiría lo mismo que en el caso para Hadoop 1.2.1 al final del documento:

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/jdk1.7.0/
```

- core-site.xml:

Para abrir el archivo:

```
hduser@hduser-VirtualBox:~$ gksudo gedit
/usr/local/hadoop/etc/hadoop/core-site.xml
```

Se añade lo siguiente entre la marca de <configuration>:

```
<property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:9000</value>
</property>
```

- yarn-site.xml:

Para abrir el archivo:

```
hduser@hduser-VirtualBox:~$ gksudo gedit
/usr/local/hadoop/etc/hadoop/yarn-site.xml
```

Se añade lo siguiente entre la marca de <configuration>:

```
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>
<property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

Para abrir el archivo:

```
hduser@hduser-VirtualBox:~$ gksudo gedit
/usr/local/hadoop/etc/hadoop/mapred-site.xml
```

Se añade lo siguiente entre la marca de <configuration>:

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
```

- hdfs-site.xml:

Para abrir el archivo:

```
hduser@hduser-VirtualBox:~$ gksudo gedit
/usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

Se añade lo siguiente entre la marca de <configuration>:

```
<property>
   <name>dfs.replication</name>
   <value>1</value>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/home/hduser/mydata/hdfs/namenode</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/home/hduser/mydata/hdfs/datanode</value>
 </property>
```

Ahora, se tendría que crear los dos directorios que especificamos en hdfs-site.xml:

```
hduser@hduser-VirtualBox:~$ sudo mkdir -p /mydata/hdfs/namenode
hduser@hduser-VirtualBox:~$ sudo chown hduser:hadoop
/mydata/hdfs/namenode
hduser@hduser-VirtualBox:~$ sudo chmod 750 /mydata/hdfs/namenode

hduser@hduser-VirtualBox:~$ sudo mkdir -p /mydata/hdfs/datanode
hduser@hduser-VirtualBox:~$ sudo chown hduser:hadoop
/mydata/hdfs/datanode
hduser@hduser-VirtualBox:~$ sudo chmod 750 /mydata/hdfs/datanode
```

Con estos cambios se tendría Hadoop 2.0 instalado. Para el trabajo se ha instalado la última versión hasta la fecha: Hadoop 2.4. Ello se puede observar en la figura 107.

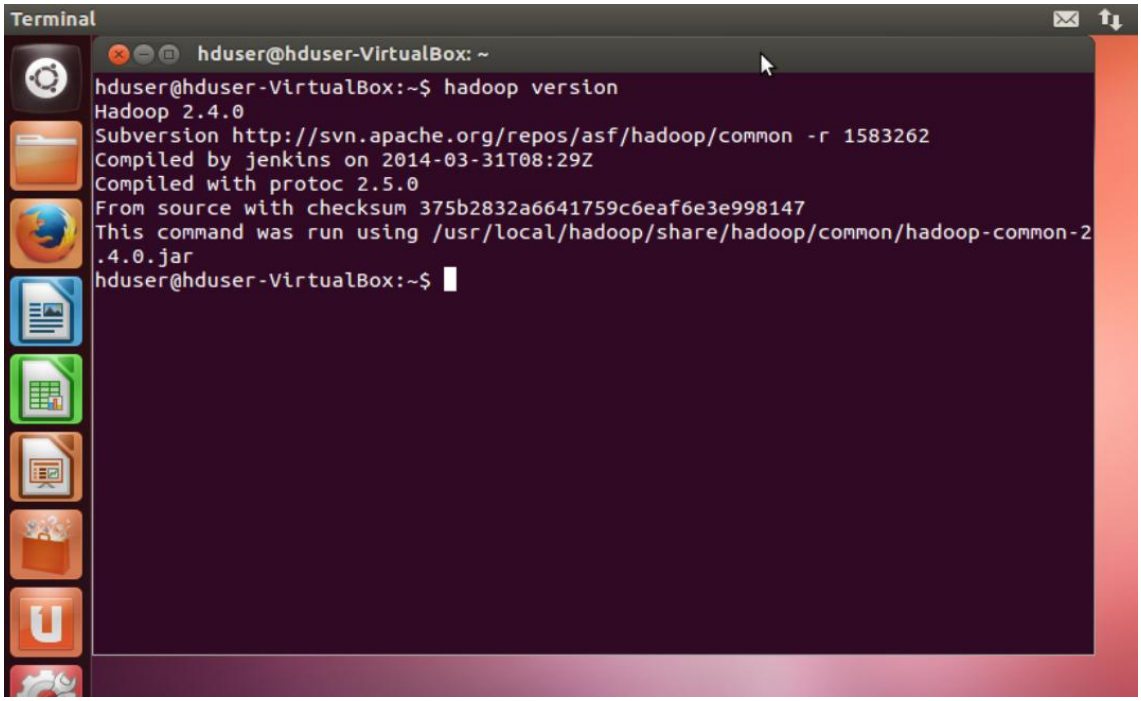


Figura 107: Hadoop 2.4 Instalado.

A.2 Principales comandos Hadoop.

En este apartado se van a presentar algunos de los comandos más utilizados para trabajar con Hadoop. Existen un amplio recopilatorio de comandos de Hadoop, el objetivo de este apartado es únicamente presentar los principales. Cuando se necesite buscar comandos más específicos será necesario consultar una referencia.

Existen gamas de comandos dependiendo del componente de Hadoop del cual necesitamos consultar información o manipular sus características. Los principales comandos son aquellos que interactúan con el HDFS. A continuación, se presentan algunos:

- mkdir: Crea un directorio en la ruta especificada.

```
hadoop fs -mkdir <paths>
```

- ls: muestra una lista de los archivos que hay en una ruta.

```
hadoop fs -ls <args>
```

- touchz: crea un archivo en la ruta especificada.

```
hadoop fs -touchz <path[filename]>
```

- cat: muestra el contenido de archivos de texto.

```
hadoop fs -cat <path[filename]>
```

- cp: copia archivos desde una fuente a un destino.

```
hadoop fs -cp <source> <dest>
```

- put: copia un archivo desde el sistema de archivos local al HDFS.

```
hadoop fs -put <source:localFile> <destination>
```

- get: copia un archivo desde el HDFS al sistema de archivos local.

```
hadoop fs -get  <source> <dest:localFileSystem>
```

- mv: mueve un archivo desde una fuente a un destino.

```
hadoop fs -mv <src> <dest>
```

- rm: elimina el archivo especificado como argumento.

```
hadoop fs -rm <arg>
```

- text: elimina toma un archivo de entrada y retorna el archivo en formato de texto.

```
hadoop fs -text <src>
```

- expung: vacía la papelera.

```
hadoop fs -expunge
```

- getmerge: toma un directorio de entrada y un archivo y concatena los archivos en el archivo local de destino.

```
hadoop fs -getmerge <src> <localdst> [addnl]
```

Otro comando muy utilizado es jar:

- jar: Ejecuta un programa contenido en un archivo jar.

```
hadoop jar <jar> [mainClass] <src> <localdst>
```

- version: Nos informa acerca de la versión de Hadoop (ver Figura 108).

```
hadoop version
```

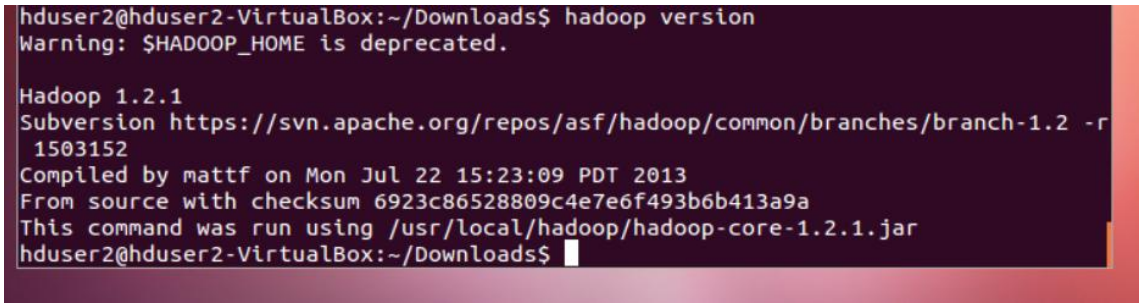


Figura 108: Comando Version en Hadoop 1.2.1

Existen muchos otros tipos de comandos con los que nos indican información sobre la configuración o sobre las tareas realizadas. Se pueden consultar en la página oficial [Ref24,WWW].