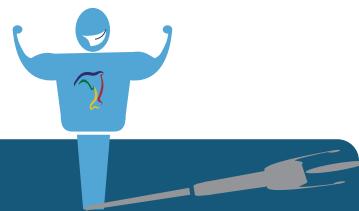


¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

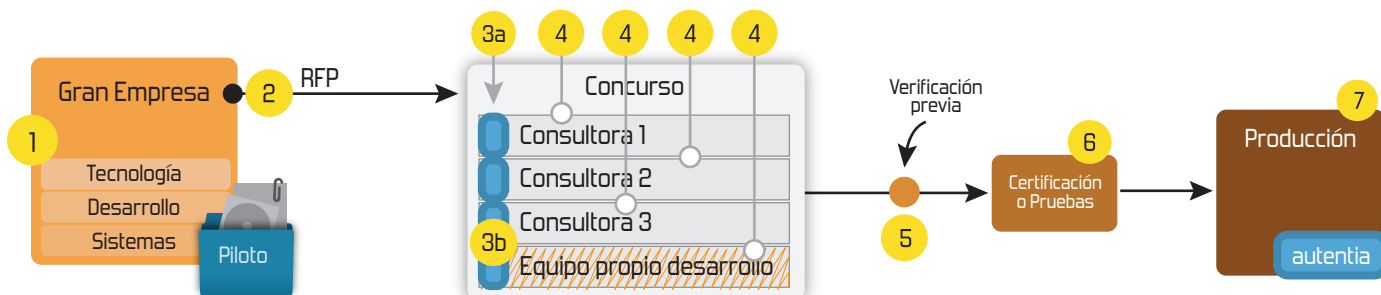
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

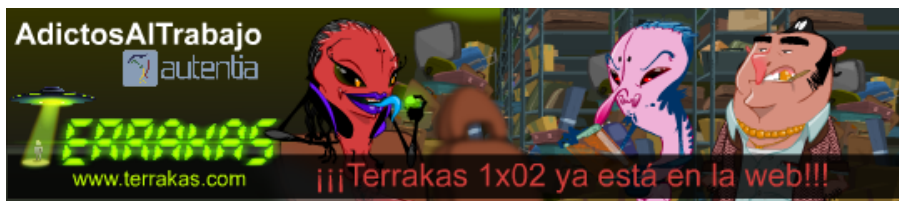
Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

Control de autenticación y
acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)



» Estás en: [Inicio](#) [Tutoriales](#) [Apache Hadoop-MapReduce](#)



Francisco Javier Martínez Páez

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Telecomunicaciones

Puedes encontrarme en [Autentia](#): Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

[Ver todos los tutoriales del autor](#)



Fecha de publicación del tutorial: 2012-02-13

Tutorial visitado 5 veces [Descargar en PDF](#)

Apache Hadoop-MapReduce

Lo primero, os dejo el enlace a los: [fuentes de este tutorial](#)

Introducción

Bienvenidos a la era de los datos. Vivimos actualmente en un mundo en el que el volumen de datos almacenados debe ser comparable a que cada ser humano del planeta tierra lleve en su bolsillo un disco duro con más información que la existente en la Biblioteca de Alejandría.

Sin embargo, mientras la capacidad de almacenar información ha aumentado drásticamente, la velocidad de acceso a esos datos almacenados en disco casi no ha variado. Entonces, parece ser que tenemos un problema, ya que el tiempo de acceso a esos datos para su análisis es excesivo para obtener resultados en un tiempo adecuado. ¿Qué se puede hacer?. Voy a atreverme a hacer unos cálculos (si hay algún error me lo perdonáis y vais al fondo de la cuestión):

- Supongamos un disco duro con una velocidad de acceso de unos 500 MB/s
- Leer un TByte nos llevaría unos 2000 segundos (una media hora).
- Supongamos ahora que tenemos la misma información (un Tera) distribuida en 100 discos (10 Gb por disco) con la misma velocidad de acceso
- Leer el tera nos llevaría unos 20 segundos (leyendo en paralelo de los 100 discos)

Por lo tanto, parece ser que el camino va por tratar de distribuir la información y poder analizarla en paralelo. Bueno, pues seguimos con la explicación:

- ¿ Distribuir la información ?
- Necesitamos un sistema de ficheros en red que nos permita leer la información sin preocuparnos de su ubicación real.
- Necesitamos además que este sistema sea tolerante a fallos y que impida pérdidas de información
- ¿ Análisis en paralelo de la información ?
- Necesitamos una nueva manera de analizar la información que permita combinar los resultados obtenidos en diferentes máquinas

Bien, pues así en pocas palabras acabo de presentar a Hadoop:

- HFDS (Hadoop Distributed Filesystem)
- Análisis mediante el modelo de programación MapReduce

Instalar Apache Hadoop

Lo primero que haremos será instalarlo. Podeis descargarlo de: [aquí](#)

Yo he descargado la versión 0.20.2. La instalación es sencilla:

- Descomprimirlo en algún lugar del disco: `tar xzf hadoop-0.20.2.tar.gz`
- Configurar algunas variables de entorno:
`HADOOP_INSTALL=/Developer/hadoop-0.20.2` (esta es mi home de hadoop)
`export PATH=$PATH:$HADOOP_INSTALL/bin`
- Necesitamos tener Java 1.6 instalada y `JAVA_HOME` configurada

Comprobamos que lo hemos instalado bien ejecutando en la consola:

```
hadoop version
```

Os tiene que mostrar algo parecido a:

Hadoop 0.20.2

Subversion https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -r 911707

Compiled by chrisdo on Fri Feb 19 08:07:34 UTC 2010

¿ MapReduce?

En pocas palabras, MapReduce es un paradigma de programación pensado para el análisis de gran cantidad de datos en paralelo. El modelo es bastante sencillo de entender, sin embargo, la capacidad de crear programas para extraer información de utilidad es otro cantar.

El modelo MapReduce se basa en el concepto de divide y vencerás, separando el procesado de la información en dos fases (adivina las fases...):

- Una fase de mapeo "Map" (que podría entenderse como una fase inicial de filtrado y agrupación de la información)
- Una fase de reducción "Reduce" (que podría entenderse como la fase de cálculo o análisis de la información obtenida en la fase anterior), o por lo menos así lo entiendo yo.

En ambos casos o fases, la entrada y salida de esas "funciones" son pares de clave-valor. Nuestro objetivo como programadores será proporcionar las funciones de Mapeo y Reducción.

En este tutorial voy a tratar únicamente de explicar el concepto de MapReduce aprovechando Hadoop (que no es el único que permite usar este paradigma de programación), y voy a dejar de lado por ahora el tema de HDFS (lo dejaré para otros tutoriales posteriores). Comento esto porque HADOOP permite tres modos de funcionamiento:

- Modo Standalone (para desarrollo). Usa una única máquina virtual (JVM)
- Pseudo distribuido (simula un cluster en la máquina local)
- Modo distribuido (cluster real de máquinas)

En el ejemplo que voy a plantear a continuación, lo ejecutaremos usando el modo Standalone, ya que únicamente deseo explicar MapReduce, por lo que no necesito por ahora ni cluster ni HDFS.

El ejemplo de MapReduce

Para explicar el concepto, vamos a usar un ejemplo sencillo (lo que se me ha ocurrido).

Voy a contar las veces que aparece una palabra en un texto. En este caso el texto es la Biblia en inglés (me he descargado los ficheros de esta página: <http://atschool.eduweb.co.uk/sbs777/bible/text/> y los he limpiado un poco, quitando información irrelevante al principio y al final de cada fichero). En concreto me he bajado los 5 libros del Génesis y el Nuevo Testamento completo (que para el ejemplo es suficiente).

Si inspeccionamos alguno de los ficheros a analizar y mostramos la información que contiene (mostramos las primeras líneas del Evangelio de Mateo):

1:1: The book of the generation of Jesus Christ, the son of David, the son of Abraham.

1:2: Abraham begat Isaac; and Isaac begat Jacob; and Jacob begat Judas and his brethren;

Podemos observar que los ficheros son de texto plano, en los que en cada línea aparece un versículo. Al principio de cada línea se indica el capítulo y el versículo al que pertenece la línea separados por ":"

Nuestro objetivo en la función de mapeo será obtener las palabras diferentes, limpiarlas un poco (quitar paréntesis, puntos, puntos y comas, comas, signos de interrogación etc...), descartar las palabras que no nos interesen y omitir la información de capítulo:versículo.

Nuestro objetivo en la función de reducción será contar.

Para empezar el desarrollo de nuestro programa, he creado un proyecto Java en Eclipse. He incluido en el classpath del proyecto todos los "jars" que vienen en la distribución de hadoop que me he descargado:

» Ap
pas
» Pa
» Mi
» Sp
» Ap
es e

Síg



Últi
emj

2011-



2011-



2011-

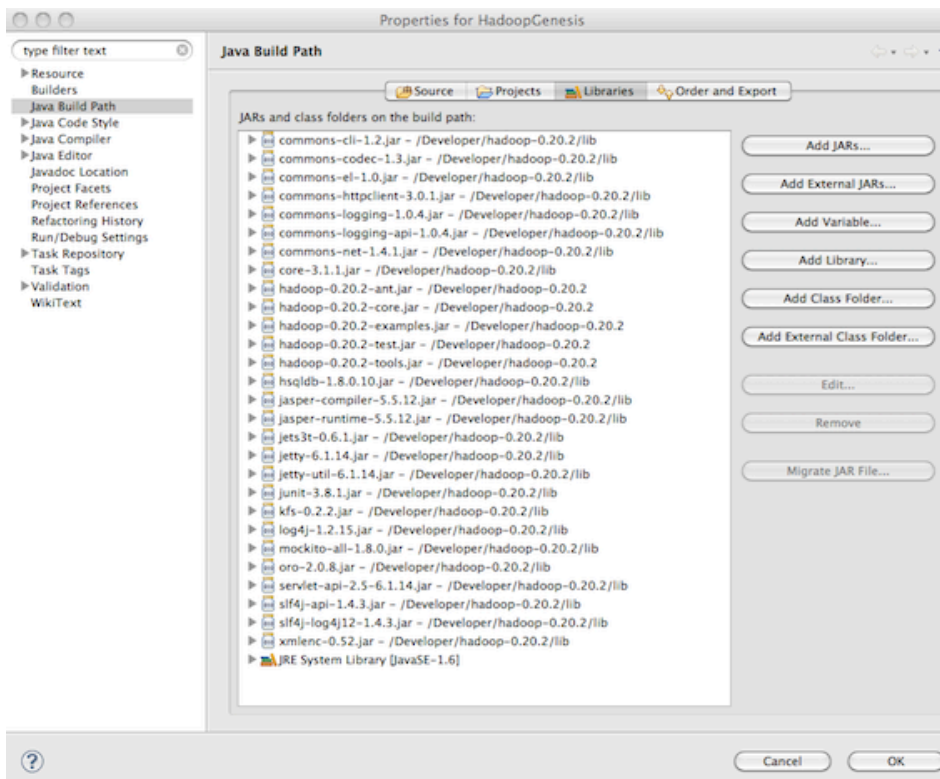


2011-



2011-





He creado una clase que será la clase que iniciará el programa: BibleWordCounter. Esta clase contendrá dos clases internas que serán las funciones de mapeo y reducción:

- BibleWordCounterMapper
- BibleWordCounterReducer

view plain print ?

```
package com.autentia.tutoriales.hadoop;
...

public class BibleWordCounter {

    static final List<String> STOP_WORDS = new ArrayList<String>();

    static {

        STOP_WORDS.add("OF");
        STOP_WORDS.add("A");
        STOP_WORDS.add("OR");
        STOP_WORDS.add("THE");
        STOP_WORDS.add("SHE");
        STOP_WORDS.add("HE");
        STOP_WORDS.add("IT");
        STOP_WORDS.add("AN");
        STOP_WORDS.add("THEY");
        STOP_WORDS.add("YOU");

    }

    static class BibleWordCounterMapper extends
        Mapper<LongWritable, Text, Text, LongWritable> {

        @Override
        protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

            String line = value.toString();

            final List<String> words = extractWordsFromLine(line);

            for (String word : words) {
                context.write(new Text(word), new LongWritable(1));
            }

        }

        private List<String> extractWordsFromLine(String line) {
            if (line != null && !"".equals(line.trim())) {
                return cleanWords(line);
            }

            return Collections.emptyList();
        }
    }
}
```

```

private boolean isChapterOrVersicle(String word) {
    char first = word.charAt(0);
    boolean startsWithNumber = true;

    try {
        Integer.valueOf(first);
    } catch (NumberFormatException e) {
        startsWithNumber = false;
    }

    return startsWithNumber && word.indexOf(':') != -1;
}

private boolean isStopWord(String word) {
    return !STOP_WORDS.contains(word);
}

private List<String> cleanWords(String line) {
    final String[] words = line.toUpperCase().split(" ");

    List<String> resultAfterCleaningWords = new ArrayList<String>(
        words.length);
    for (String word : words) {
        if (!"".equals(word) && !isChapterOrVersicle(word)) {

            String finalWord = cleanWord(word);

            if (!isStopWord(word)) {
                resultAfterCleaningWords.add(finalWord);
            }
        }
    }
    return resultAfterCleaningWords;
}

private String cleanWord(String word) {
    String finalWord = word;
    if (word.endsWith(".") || word.endsWith(",") || word.endsWith(";")
        || word.endsWith(":") || word.endsWith("(")
        || word.endsWith("?") || word.endsWith("!")) {
        finalWord = cleanWord(word.substring(0, word.length() - 1));
    }

    if (word.startsWith("(") {
        finalWord = cleanWord(word.substring(1, word.length()));
    }

    return finalWord;
}

}

static class BibleWordCounterReducer extends
    Reducer<Text, LongWritable, Text, LongWritable> {

    @Override
    protected void reduce(Text key, Iterable<LongWritable> values,
        Context context) throws IOException, InterruptedException {

        long timesThisWord = 0;

        for (LongWritable value : values) {
            timesThisWord++;
        }

        context.write(key, new LongWritable(timesThisWord));
    }
}

public static void main(String[] args) throws Exception {

    long inicio = System.currentTimeMillis();

    if (args.length != 2) {
        System.err
            .println("Modo de uso: BibleWordCounter <path entrada> <path salida>");
        System.exit(-1);
    }
    Job job = new Job();
    job.setJarByClass(BibleWordCounter.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(BibleWordCounterMapper.class);
}

```

```

    job.setReducerClass(BibleWordCounterReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);

    boolean ok = job.waitForCompletion(true);

    System.out.println("Ejecutado en:"
        + (System.currentTimeMillis() - inicio) + " ms");

    System.exit(ok ? 0 : 1);

}

}

```

Nos fijaremos primero en la función "protected void map(LongWritable key, Text value, Context context)" (función de mapeo).

Esta función será invocada una vez por cada línea existente en los ficheros de datos de entrada. En "key" nos llegará el número de línea (no nos interesa). En "value" nos llegará el texto de cada línea.

Si seguís el código detenidamente, lo único que hacemos es extraer en una lista las palabras de cada línea (con las características explicadas anteriormente) y para cada palabra "emitir" un par clave-valor compuesto por "palabra"-1

Por lo tanto, para el ejemplo mostrado anteriormente (y si únicamente tuviéramos estas dos líneas):

1:1: The book of the generation of Jesus Christ, the son of David, the son of Abraham.
 1:2: Abraham begat Isaac; and Isaac begat Jacob; and Jacob begat Judas and his brethren;

Emitiremos:

(BOOK,1) (GENERATION,1) (JESUS,1) (CHRIST,1) (SON,1) (DAVID,1) (SON,1) (ABRAHAM,1)
 (ABRAHAM,1) (BEGAT,1) (ISAAC,1) (ISAAC,1) (BEGAT,1) (JACOB,1) (JACOB,1) (BEGAT,1) (JUDAS,1) (HIS,1)
 (BRETHREN,1)

Una vez emitidos estos valores, antes de invocar la función de reducción, el framework de MapReduce (Hadoop) ordenará y agrupará por claves, de la siguiente manera:

(ABRAHAM,[1,1]) (BEGAT,[1,1,1]) (BRETHREN,[1]) (BOOK,[1]) ...

Por lo tanto, a la función de reducción, la información le llegará de la manera mostrada.

Fijémonos ahora en la función de reducción: "protected void reduce(Text key, Iterable values, Context context)". Esta será invocada una vez por cada clave diferente emitida en la función de mapeo.

Recibirá en "key" la clave emitida y en "value", la lista de valores emitidos para esa clave.

Por lo tanto, nuestro trabajo será iterar y "contar" las veces que iteramos. El valor resultante será el que emitiremos.

Si seguimos el ejemplo, emitiremos:

(ABRAHAM,2) (BEGAT,3) (BRETHREN,1) (BOOK,1) ...

Pues ya tenemos el ejemplo preparado. En la función main lo único que hacemos es lanzar el programa indicando los parámetros necesarios para su ejecución.

Para ejecutar el programa, abrid una ventana de comandos. Situaos en la raíz del proyecto que os habéis creado en Eclipse y escribid:

```
export HADOOP_CLASSPATH=./bin
```

Para configurar el classpath y que encuentre nuestra clase.

Ahora escribid: *hadoop com.autentia.tutoriales.hadoop.BibleWordCounter /Trabajo/tutoriales/hadoop/data /ejemplo*

Invocamos a hadoop indicando la clase que contiene el main(). El primer parámetro es el directorio que contiene la información a analizar y el segundo parámetros es el directorio donde exportará los resultados (no debe existir, si existe os dará un error).

Una vez terminado, podemos ver los resultados. Vamos a buscar la palabra CHRIST:

```
CHRIST 525
CHRIST'S 16
CHRISTIAN 2
CHRISTIANS 1
CHRISTS 2
CHRYSOLITE 1
CHRYSOPRASUS 1
CHURCH 70
CHURCHES 35
```

Conclusiones

Si nos quitamos las gafas y echamos una mirada un tanto miope, podríamos decir... pero Paco, si esto lo puedo hacer con una shell script y awk o con un programa java (o cualquier lenguaje que maneje ficheros y cadenas de caracteres); y yo diría sí, efectivamente, pero entonces os tendría que remitir a la introducción del tutorial, porque la gran diferencia está en que este programa se puede paralelizar tanto en lectura de información como en ejecución (así que si se te ha ocurrido esta idea...ya sabes, vuelve a leer la introducción)

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

Por favor, vota +1 o compártelo si te pareció interesante

Share |

0

¿Te gusta adictosaltrabajo.com? Síguenos a través de:



Anímate y coméntanos lo que pienses sobre este TUTORIAL:

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Regístrate** y accede a esta y otras ventajas «



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

IMPULSA

Impulsores

Comunidad

[¿Ayuda?](#)

sin clicks

0 personas han traído clicks a esta página

+

+

+

+

+

+

+

+

powered by [karmacracv](#)

Copyright 2003-2012 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) |

W3C

XHTML 1.0

W3C

CSS

XML

RSS

XML

ATOM