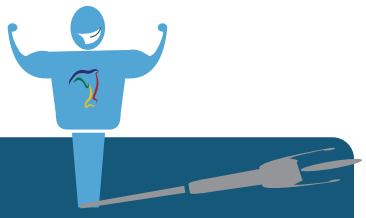
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

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

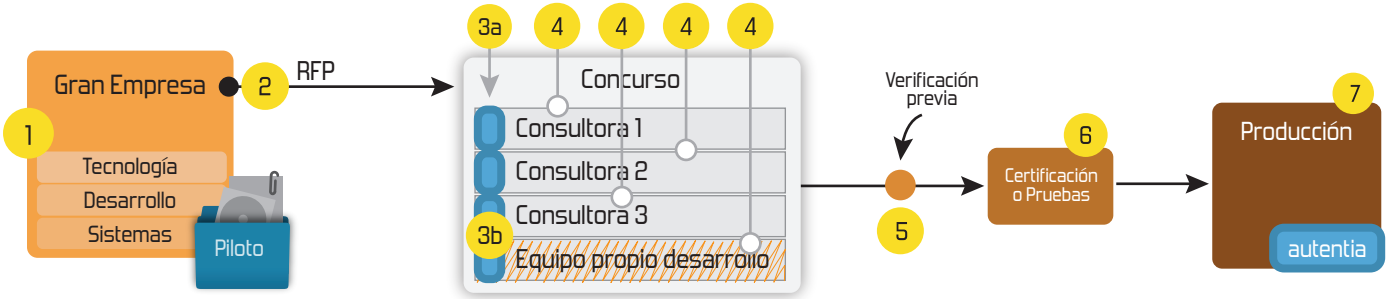
1. Desarrollo de componentes y proyectos a medida



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
- 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
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

AdictosAlTrabajo.com
"¡La primavera ha venido, nadie sabe cómo ha sido!"

autentia
Soporte a desarrollo informático

Entra en Adictos a través de
E-mail
Contraseña
Entrar
Registrarme
Olvidé mi contraseña

Inicio Quiénes somos Formación Comparador de salarios Nuestros libros Más

» Estás en: Inicio Tutoriales Introducción a Apache Hive, el datawarehouse de Hadoop

Juan Alonso Ramos

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software

Puedes encontrarme en Autentia: Ofrecemos de servicios soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

FARNELL.COM/ES
uscador de productos ¡Es gratis! - La nueva herramienta ...

Fecha de publicación del tutorial: 2015-03-25

Tutorial visitado 1 veces Descargar en PDF

Patrones de diseño en Hadoop: Patrón Partitioner

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. Introducción al patrón Partitioner
- 4. Implementación del patrón
- 5. Componentes básicos
- 6. Componentes avanzados
- 7. Referencias
- 8. Conclusiones

1. Introducción.

El objetivo de un patrón de diseño de software es el de buscar una solución a un problema común, una vez encontrada debe ser probada su efectividad para que pueda considerarse una buena práctica la utilización de dicho patrón y así poder ser estandarizado.

En la mayoría de aplicaciones podemos identificar problemas similares, las aplicaciones MapReduce no iban a ser menos por lo que también se producen problemas que merecen ser estudiados. Este tutorial está basado en los patrones de diseño del libro MapReduce Design Patterns.

En este tutorial vamos a ver un ejemplo de cómo aplicar el patrón **Partitioner** que pertenece al grupo de patrones de organización de datos.

2. Entorno.

El tutorial se ha realizado con el siguiente entorno:

- Ubuntu 12.04 64 bits
- Oracle Java SDK 1.6.0_27
- Apache Hadoop 2.2.0
- Apache Maven 3.1.1

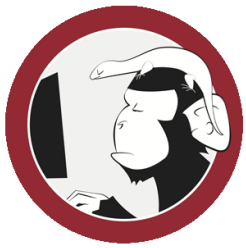
3. Introducción al patrón Partitioner

El patrón Partitioner (o de distribución o particionado) indica que debemos dividir en grupos los datos similares para ser tratados en conjunto. La división para el tratamiento de los datos se debe hacer de forma lógica, estudiando los datos previamente y estableciendo un criterio de manera que la distribución quede uniforme.

En tareas MapReduce el resultado de la ejecución completa depende de que se hayan procesado todos los trabajos de todos los nodos del cluster. Es por ello que la distribución de los datos debe ser lo más uniforme posible ya que el resultado no estará listo hasta que terminen todas las tareas Reduce. Si uno de los reducers recibe la mayoría de los datos tendremos un cuello de botella en el sistema. Esto se soluciona aplicando el patrón 'Partitioner'. Una vez terminada la fase de mapeo, se llamará al 'Partitioner' que decidirá a que 'Reducer' invocar con los datos de salida de la tarea anterior. Una vez que apliquemos este patrón el fichero de los datos de salida quedará dividido según el número de particiones que tengamos.

El particionado por fecha es uno de los esquemas más típicos cuando se trabaja con un conjunto de datos que disponga de este dato ya que es común que los datos estén bien distribuidos por este campo aunque no siempre puede ocurrir. En una distribución de población otros criterios pueden ser por país, región, sexo, edad, etc.

Catálogo de servicios Autentia



Síguenos a través de:

Últimas Noticias

» 2015: ¡Volvemos a la oficina!

» Curso JBoss de Red Hat

» Si eres el responsable o líder técnico, considérate desafortunado. No puedes culpar a nadie por ser gris

» Portales, gestores de contenidos documentales y desarrollos a medida

» Comentando el libro Start-up Nation, La historia del milagro económico de Israel, de Dan Senor & Salu Singer

Histórico de noticias

Últimos Tutoriales

» Movimientos de la cámara

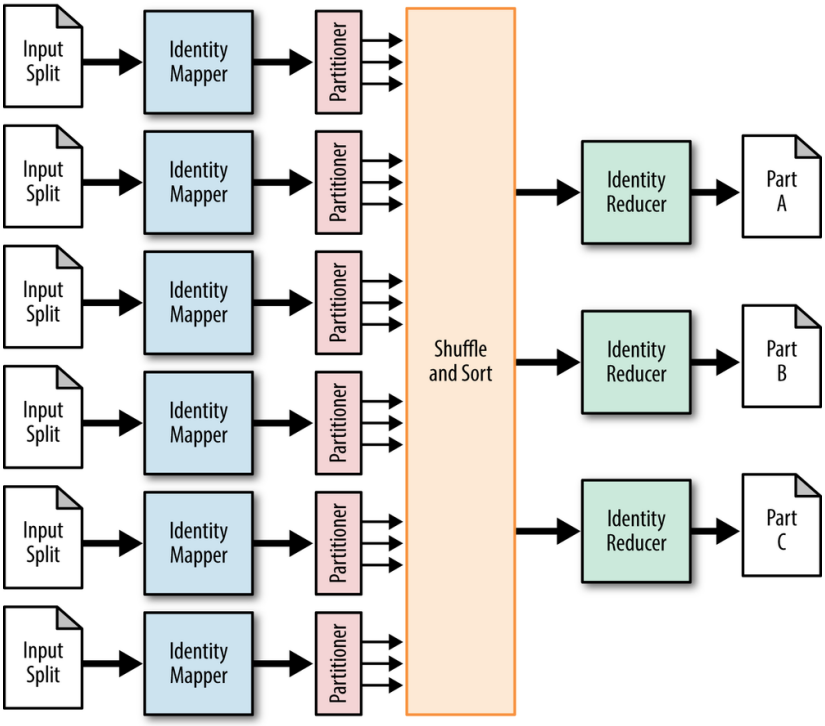
» Introducción a Apache Hive, el datawarehouse de Hadoop

» Mixins en Java y Java8 !Sí, es posible!

» Introducción a la Sandbox HDP - Hortonworks Data Platform

» Configura e interpreta las métricas de Sonarqube para conocer la calidad de tu código

El funcionamiento del 'Partitioner' se muestra en la siguiente imagen. Tras la fase de mapeo y previa al 'Suffle and Sort' se ejecutará el 'Partitioner' definido.



Fuente: MapReduce Design Patterns O'Reilly

4. Implementación del patrón

Como en otras ocasiones, voy a utilizar una fuente de datos pública para un caso práctico de aplicación del patrón 'Partitioner'. En este caso he escogido el censo de población de Madrid a fecha del enero de 2015 clasificado por distritos y barrios. En el CSV aparece lo siguiente:

```
"COD_DISTRITO";"DESC_DISTRITO";"COD_DIST_BARRIO";"DESC_BARRIO";"COD_BARRIO";"COD_DIST_S
1;CENTRO;101;PALACIO;1;1001;1;0;3;3;2;
1;CENTRO;101;PALACIO;1;1001;1;1;5;3;1;1
1;CENTRO;101;PALACIO;1;1001;1;2;8;3;1;
...
```

En el fichero CSV se muestra una línea por cada edad de las personas, desde los 0 años en adelante clasificada por distrito y barrio de Madrid mostrando el número de personas españolas y extranjeras, hombres y mujeres. Con este dataset podríamos hacernos una idea por ejemplo del número de extranjeros que viven en cada barrio, qué barrio tiene más niños menores de 10 años o el barrio con más pesonas por encima de los 80 por poner algún ejemplo. Todo sería cuestión de jugar con los datos, lo importante es disponer de ellos lo cual agradecemos que estén disponibles de forma pública.

Puedes descargar el CSV desde el portal de datos abiertos del Ayuntamiento de madrid, enlace.

Mapper

Para aplicar el patrón 'Partitioner' vamos a recoger los datos de la edad y el número de personas que están censadas, hombres y mujeres. Haremos un Writable propio para guardar todos los datos. El Mapper emitirá como clave el distrito.

```
public static class AgeMapper extends Mapper<Object, Text, Text, AgeWritable> {

    @Override
    protected void map(Object key, Text value, Context context) throws IOException, Inte

        final String[] tokens = value.toString().split(";");
        final String descDistrito = tokens[1];
        final String age = tokens[7];
        final String numHombres = tokens[8];
        final String numMujeres = tokens[9];
        final String numHombresExt = tokens[10];
        final String numMujeresExt = tokens[11];
        final String keyValue = descDistrito;
        final AgeWritable ages = new AgeWritable(age, numHombres, numMujeres, numHombres

        context.write(new Text(keyValue), ages);
    }
}
```

Reducer

El Reducer en este caso se encargará de sumar el número de personas ya que únicamente vamos a sacar el cómputo general de personas censadas por distrito de Madrid independientemente de su edad, sexo o si son españoles o extranjeros.

```
public static class AgeReducer extends Reducer<Text, AgeWritable, Text, IntWritable>

    @Override
    protected void reduce(Text key, Iterable<AgeWritable> values, Context context) thr
```

Últimos Tutoriales del Autor

» Introducción a Apache Hive, el datawarehouse de Hadoop

» Introducción a la Sandbox HDP - Hortonworks Data Platform

» Cómo crear un mapa interactivo en CartoDB

» Crea interfaces web amigables con Twitter Bootstrap

» Cómo implementar una nube de etiquetas con D3.js

```
        int total = 0;

        for (AgeWritable age : values) {
            total += NumberUtils.toInt(age.getNumMales()) + NumberUtils.toInt(age.getN
        }

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

Partitioner

El Partitioner se encargará de distribuir los datos de la población por edades. Imagina que tenemos 4 procesos de reducción, dividiremos las edades en 4 grupos en el Partitioner para que la fase de reducción sea lo más equitativa posible y trabaje de forma óptima.

```
public class AgePartitioner extends Partitioner<Text, AgeWritable> {

    @Override
    public int getPartition(Text key, AgeWritable ageWritable, int partitions) {
        if (partitions == 0) {
            return 0;
        }

        final int age = NumberUtils.toInt(ageWritable.getAge());

        if (age <= 25) {
            return 0;
        } else if (age > 25 && age <= 50) {
            return 1;
        } else if (age > 50 && age <= 75) {
            return 2;
        } else {
            return 3;
        }
    }
}
```

Driver

Para juntar todas las piezas tenemos el Driver donde configuramos el Job indicando el Mapper, el Reducer, el Partitioner, el formato de los datos de salida del Mapper y del Reducer y el número de tareas reducer disponibles. Lógicamente el número de reducers vendrá condicionado a las máquinas que formen el clúster donde correrá el proceso MapReduce. En local si no disponemos más de una no veremos más que un Reducer.

```
public class AgeManager extends Configured implements Tool {

    @Override
    public int run(String[] args) throws Exception {

        if (args.length != 2) {
            System.err.println("AgeManager required params: {input file} {output dir}");
            System.exit(-1);
        }

        deleteOutputFileIfExists(args);

        final Job job = new Job(getConf(), "ageManager");

        job.setJarByClass(AgeManager.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

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

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(AgeWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(AgeMapper.class);
        job.setReducerClass(AgeReducer.class);

        job.setPartitionerClass(AgePartitioner.class);
        job.setNumReduceTasks(4);

        job.waitForCompletion(true);

        return 0;
    }

    private void deleteOutputFileIfExists(String[] args) throws IOException {
        final Path output = new Path(args[1]);
        FileSystem.get(output.toUri(), getConf()).delete(output, true);
    }

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new AgeManager(), args);
    }
}
```

El resultado tras la ejecución del proceso queda así:

```
ARGANZUELA  150467
BARAJAS 45812
CARABANCHEL 240853
CENTRO  131568
CHAMARTIN   141738
CHAMBERI    137767
CIUDAD LINEAL   212264
```

```
FUENCARRAL-EL PARDO 233341
HORTALEZA   174640
LATINA  234839
MONCLOA-ARAVACA 116280
MORATALAZ   95031
PUENTE DE VALLECAS  226744
RETIRO  118212
SALAMANCA   142843
SAN BLAS-CANILLEJAS 152780
TETUAN  151313
USERA   133035
VICALVARO   69460
VILLA DE VALLECAS   99702
VILLAVERDE  140974
```

6. Conclusiones.

Después de lo que hemos visto es muy importante definir correctamente la función que se encarga del particionado de los datos si queremos que el rendimiento del algoritmo MapReduce sea óptimo.

Si no tenemos clara la función que obtiene partes de los datos equivalentes podemos utilizar una función aleatoria de manera que la distribución de los datos sea equitativa.

Espero que te haya sido de ayuda.

Un saludo.

Juan

A continuación puedes evaluarlo:

Regístrate para evaluarlo

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

Share |

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

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

Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

Copyright 2003-2015 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto