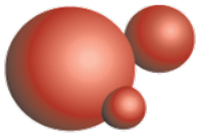


QUALITY OBJECTS

C/Santa Leonor Nº65
Edificio C- 2ª Izda
28037 MADRID
Tel.: +34 914 066 890
Fax: +34 914 066 891
http://www.qualityobjects.com

RecommendInCloud: Motor de recomendaciones para portales de e-comerce en hServer

Preparado por:	Rubén Lajo
Versión:	1.0
Fecha:	13 de Enero de 2016

Personas de contacto

Para posibles aclaraciones y o dudas sobre el contenido y alcance del presente documento, las personas de contacto será:

César Carralero
E-mail ccarralero@qualityobjects.com

Rubén Lajo
E-mail rlajo@qualityobjects.com

1 INTRODUCCIÓN

A la hora de enfrentarnos a una elección entre una de varias opciones, siempre es útil obtener una recomendación de alguien que ha realizado previamente la misma decisión ya que cuantas más fuentes de información tengamos alrededor de una elección, mayor probabilidad de acierto tendremos a la hora de enfrentarnos a ella.

Con el auge de Internet y del comercio electrónico el número de posibilidades sobre las que podemos elegir crece de tal forma que es imposible que una única persona o conjunto de personas sean capaces de ofrecernos una recomendación para cada una de las posibles opciones. Para enfrentarse a este problema surgen las técnicas llamadas de inteligencia colectiva.

La inteligencia colectiva es una forma de inteligencia que surge de la colaboración y concurso de un gran conjunto de individuos. Con la Web 2.0 aparecen nuevas formas de relacionarse, en las que los consumidores pasan a ser también creadores como consecuencia de una gran facilidad para la aportación de información. De esta forma se consigue obtener una base de conocimiento distribuida, actualizada y coordinada en tiempo real en la que todos los individuos participan y se pueden beneficiar.

En este documento se va a describir la forma en que se ha implementado el motor de recomendaciones *RecommendInCloud* haciendo uso de tecnologías *Cloud* y *BigData*.

Las características principales del motor construido son:

- Se trata de un sistema autónomo, es decir, una vez puesto a funcionar no necesita de intervención. Se alimenta del comportamiento de los usuarios y basa sus recomendaciones en éste.

- Es un sistema independiente de los productos. Acepta entradas de preferencias de usuarios por ítems y recomienda ítems a usuarios independientemente de lo que esos ítems signifiquen para el usuario final. Funciona de igual forma cuando un ítem representa una noticia, un producto de una tienda online, una entrada en un blog, un viaje...

En el resto del documento se van a tratar los siguientes puntos:

- Descripción funcional: Detalle de la funcionalidad proporcionada por el motor.
- Arquitectura: Descripción técnica de los componentes del sistema.
- Prestaciones: Estudio del rendimiento esperado por el motor bajo grandes cargas.
- Conclusiones y posibles trabajos futuros.

2 DESCRIPCIÓN FUNCIONAL

En este apartado se va a describir la funcionalidad proporcionada por *RecommendInCloud*.

- *RecommendInCloud* proporciona un servicio *Cloud* de recomendaciones basado en análisis *BigData.* Permite a todo tipo de portales de comercio electrónico beneficiarse de las ventajas que un motor inteligente de recomendaciones puede aportar al aumento de ventas y fidelización de clientes.

- *RecommendInCloud* está diseñado como una solución en modo Cloud Computing de forma que cualquier portal del mundo puede hacer uso del mismo como Software as a Service (SaaS) de forma sencilla, realizando llamadas REST con una API Javascript, desde su portal de comercio electrónico al servicio de recomendaciones.

- *RecommendInCloud* es un sistema de recomendaciones real, basado en Inteligencia Colectiva. Es decir, el sistema analiza de forma transparente a los usuarios, su comportamiento en el portal, qué compran, en qué se interesan, etc; y utiliza ese conocimiento para recomendar productos a otros usuarios. Se retroalimenta constantemente con todas las actuaciones de los usuarios para mejorar la calidad de las recomendaciones.

- Es eficiente, porque es capaz de encontrar productos recomendados que realmente sean significativos para los usuarios. No se trata de recomendar lo que el sitio de *eCommerce* quiera vender (aunque esto también es posible), sino lo que los usuarios realmente quieran comprar. Para ello utiliza algoritmos que permiten encontrar usuarios con gustos parecidos y analiza qué productos son los más indicados para recomendar a cada persona.

- Es independiente de los productos a recomendar. No analiza qué es cada producto, no tiene prerequisitos, únicamente analiza lo que realmente les interesa a los usuarios. Por lo tanto a un usuario de un viaje, le puede recomendar otro viaje que le pueda interesar, o un juego de maletas a comprar, siempre que ese comportamiento de compra se haya producido más veces por otros usuarios.

- Es independiente del perfil de las personas. No necesita conocer a las personas para analizar sus perfiles de compra. No necesita conocer donde viven, cuanto ganan o que le digan sus intereses. Analiza únicamente su comportamiento y realiza la recomendación de forma transparente al usuario, encontrando productos que realmente le interesan.

- Dado que es independiente del perfil de usuario y de las características del producto, Es valido para cualquier tipo de portal de *eCommerce*, incluso para portales de viajes, inmobiliarios, de búsqueda de empleo, foros, blogs, buscadores, etc.

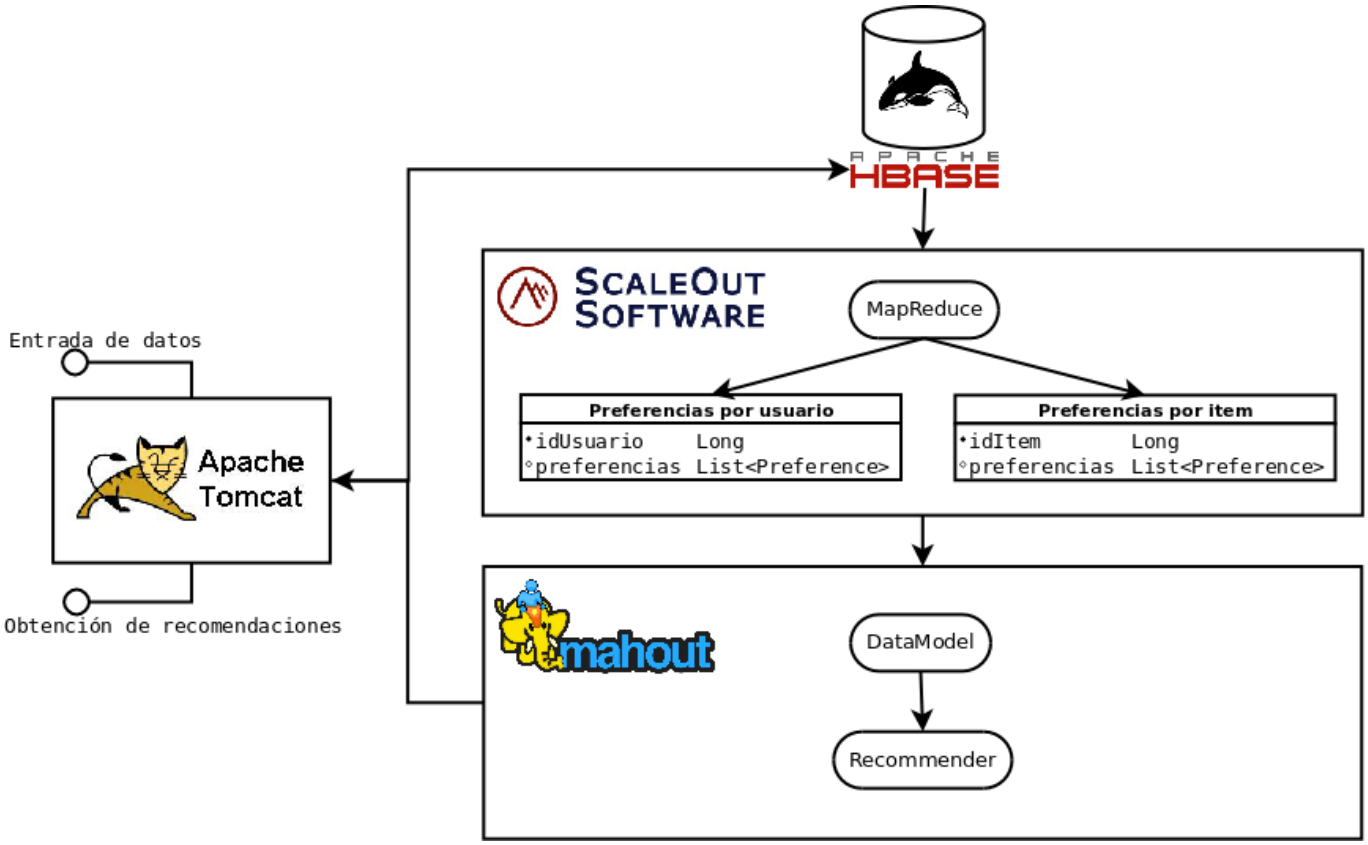
- Soporta doble motor de recomendaciones:
  - Quien se intereso en ítem1, también se intereso en ítem2. Por ejemplo, quien visualizó un teléfono móvil, también visualizó otro de similares características.
  - Quien compro ítem1, también compro ítem2. Por ejemplo, quién compró un teléfono móvil, también compró una funda y un protector de pantalla para el mismo.

- *RecommendInCloud* proporciona una interfaz REST que permite a los portales integrados la interacción con el motor mediante un sencillo API en javascript. Las operaciones disponibles serán:
  - addPreference: Permite indicar la preferencia de un usuario por un ítem.
  - getRecommendation: Obtener las recomendaciones relacionadas con un ítem.

- addAndGetRecommendation: Unifica las dos operaciones anteriores. Almacena la preferencia del usuario por un ítem y además devuelve las recomendaciones solicitadas para dicho ítem.

3 ARQUITECTURA

En el siguiente esquema se muestra una visión general del motor de recomendaciones.



Este desglose en componentes ofrece una serie de ventajas.

En primer lugar obtenemos un sistema tolerante a fallos, en caso de que el sistema se caiga tenemos toda la información de las preferencias en la base de datos. Como veremos más adelante bastaría iniciar los procesos *MapReduce* para restaurar el sistema en el punto anterior al fallo.

Aparte de lo anterior, al hacer uso del software de *ScaleOut* obtenemos la gran potencia de su *DataGrid* en memoria. Esto es una enorme ventaja sobre otros sistemas de recomendación basados en *Hadoop* ya que toda la información necesaria para realizar la recomendación se encuentra en memoria en todo momento, no es necesario ir a disco o a una BD a por ella con la ventaja en rendimiento que esto implica.

Otra gran ventaja del sistema implementado es la integración de *Mahout* sobre *ScaleOut*. Aprovechamos la enorme base de algoritmos ya creados en *Mahout* con el trabajo en memoria de *ScaleOut*.

El proyecto se ha implementado como un proyecto multi modular Maven con los siguientes módulos:

- recommend-common: Funcionalidad común, capa de acceso a datos, servicios, etc.
- recommend-web: Aplicación web que proporciona la interfaz REST para permitir el acceso al motor de recomendaciones desde los portales eComerce.
- recommend-scaleout: Funcionalidad relacionada con ScaleOut. Este proyecto permite ser ejecutado en forma de proceso Batch para el cálculo de los MapReduce.
- recommend-clientapp: Aplicación web con un cliente de ejemplo de uso de la interfaz.

3.1 Aplicación web (recommend-web)

Se trata de una aplicación web J2EE que se puede desplegar en cualquier servidor que cumpla la especificación de Servlets J2EE. Su principal cometido es proporcionar la interfaz REST para el acceso al motor de recomendaciones. Además proporciona un fichero javascript para facilitar a las aplicaciones cliente el acceso a esta interfaz.

Para la implementación de la interfaz REST se ha usado el framework *Spring MVC* y para la serialización de objetos de entrada y salida se ha usado la librería *Jackson*.

En *Spring MVC*, para que un método de un controlador sea expuesto al acceso mediante REST debe encontrarse dentro de una clase con la anotación *@RestController*:

```
@RestController
@RequestMapping("/recommend")
public class RecommendController {
```

Una definición de este tipo mapeará todas las peticiones al namespace */recommend con* este controlador.

3.1.1 Operaciones

Como hemos visto vamos a ofrecer tres operaciones:

- *addPreference*: Recibe como entrada un usuario, el identificador de un ítem y un tipo de operación (compra o visualización). Inserta en *HBase* la preferencia del usuario por el ítem.
- *getRecommendation*: Recibe como parámetros de entrada un ítem, el tipo de operación y el número de recomendaciones deseadas. Devuelve las recomendaciones solicitadas para el ítem dado.
- *addAndGetRecommendation*: Unifica las dos operaciones anteriores. Recibe como parámetros de entrada el usuario, el ítem, el tipo de operación y además el número de recomendaciones deseadas. Realiza la inserción en *HBase* y además los ítems recomendados.

3.1.2 Formato de los datos

Teniendo en cuenta que las llamadas REST se van a realizar desde navegadores web en distintos entornos, el formato de intercambio de los datos debe ser JSONP (JSON with Pading https://en.wikipedia.org/wiki/JSONP). Este formato es usado para evitar las restricciones de seguridad de llamadas entre dominios (*Cross-Site Requests*) que implementan la mayor parte de los navegadores actuales y que impiden realizar llamadas AJAX desde una página en un dominio a un servidor que se encuentre en otro dominio.

La solución a esta restricción consiste en cargas de scripts cuyo resultado es la llamada a una función javascript local enviándole los datos resultantes como parámetro. Por ejemplo, una llamada a *getRecommendation* podría devolver la siguiente cadena JSON:

```
[{"itemID":1,"value":0.69416445},{"itemID":73,"value":0.68052053},
{"itemID":11,"value":0.6616981}];
```

Usando JSONP la llamada devolvería:

```
Recommender.processGetRecommendations([{"itemID":1,"value":0.69416445},
{"itemID":73,"value":0.68052053},{"itemID":11,"value":0.6616981}]);
```

El resultado sería la invocación a la función local *Recommender.processGetRecommendations* dentro de la cual podríamos procesar el resultado.

Una vez definido el formato de los datos, la implementación de los métodos del controlador sería:

```
@RequestMapping(value = "getRecommendations", method = RequestMethod.GET)
public MappingJacksonValue getRecommendations(@RequestParam String callback,
      @RequestParam Long itemId,
      @RequestParam RequestType type,
      @RequestParam(defaultValue = "3") Integer howMany,
      HttpServletResponse response) {
```

Esta declaración dirigirá las peticiones GET al contexto */recommend/getRecommendations* hacia este método.

3.1.3 Javascript para los clientes

Para facilitar el acceso a estas funciones por parte de las aplicaciones cliente, se ha creado un fichero javascript que permite ser importándolo en las páginas de las aplicaciones cliente y facilita el acceso a esta interfaz.

Para la implementación de este fichero se ha usado el patrón *Revealing Module Pattern* que permite definir todas las funciones dentro de un namespace y que no entren en conflicto con las funciones que defina el cliente en sus propios js.

En concreto se han definido funciones para la llamada a cada una de las operaciones vistas anteriormente, esto es:

- Recommender.addPreference(userId, itemId, type)
- Recommender.getRecommendations(itemId, type)
- Recommender.addAndGetRecommendations(userId, itemId, type)

Además se exige que el cliente sobrescriba las siguientes funciones para procesar los resultados devueltos en la llamada (*callbacks*):

- Recommender.processAddPreference(data) – data puede ser la cadena “OK” si todo ha ido bien o una descripción de un error en caso de que se produzca.
- Recommender.processGetRecommendations(data) – data va a ser una matriz de objetos con las propiedades itemID y value.
- Recommender.processAddAndGetRecommendations(data) – data va a ser una matriz de objetos con las propiedades itemID y value.

3.2 HBase

Debemos almacenar de forma persistente los datos de interacción de los usuarios. Para ello usaremos una base de datos noSQL que nos proporcione un acceso rápido y ágil a los datos. En nuestro caso usaremos la base de datos *HBase*.

Para el almacenamiento de datos en *HBase* se ha definido la tabla *itemRequest* con las siguientes columnas:

- userId: Identificador del usuario que hace la petición
- itemId: Identificador del ítem solicitado
- type: Tipo de solicitud. Valores posibles: *BUY* para compra, *VIEW* para visualización
- timestamp: Instante de grabación de la entrada

Se ha creado un test *JUnit* que realiza la creación de la tabla e inserta un conjunto de datos de prueba. Para ejecutarlo mediante Maven basta con ejecutar el comando:

```
mvn test HbaseTest
```

Para el acceso a la base de datos desde java se ha creado un objeto DAO (*ItemRequestDaoImpl*) que se apoya en las librerías *Apache HBase Client* y en la parte correspondiente a *HBase* de la librería *Spring for Apache Hadoop* del proyecto *Spring Data*. El elemento más importante de esta última es la clase *HbaseTemplate*. Este bean lo definimos en el fichero *applicationContext.xml:*

```
<hdp:configuration />

<hdp:hbase-configuration zk-quorum="${hbase.host}" zk-port="${hbase.port}"
delete-connection="false"/>

<bean id="hbaseTemplate"
class="org.springframework.data.hadoop.hbase.HbaseTemplate">
      <constructor-arg ref="hbaseConfiguration"/>
</bean>
```

La configuración de la base de datos se ha sacado al fichero *application.properties* en el cual se puede definir la localización del servidor mediante las variables *hbase.host* y *hbase.port*.

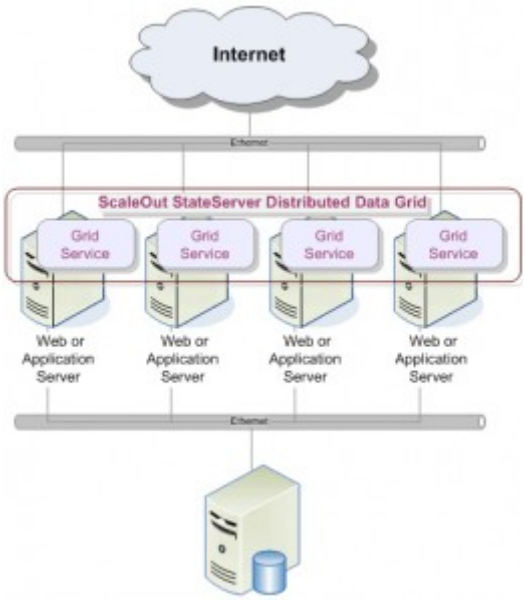
3.3 ScaleOut

Scaleout provee soluciones *In-Memory* para el almacenamiento y procesamiento de grandes cantidades de datos. En el caso de *RecommendInCloud* se han usado dos productos principalmente:

- ScaleOut StateServer: Proporciona el IMDG (In-Memory Data Grid), una solución distribuida para el almacenamiento en memoria de grandes cantidades de datos.
- ScaleOut hServer: Permite la ejecución de algoritmos MapReduce sobre el IMDG. Es compatible con trabajos programados para Apache Hadoop.

3.3.1 ScaleOut StateServer: Almacenamiento In-Memory

StateServer es la base de la infraestructura de *ScaleOut*. Proporciona un sistema conocido como IMDG. Se trata de un almacenamiento en memoria distribuido entre varios servidores. Gestiona de forma automática tanto la distribución de los datos como la tolerancia a errores.



ScaleOut proporciona al programador un sencillo API para el acceso al IMDG. En el caso de *RecommendInCloud* se va a usar el IMDG para el almacenamiento de las preferencias de los usuarios clasificadas en la forma que resulte más óptima para el recomendador. A continuación se muestra la sencilla forma de acceso a uno de los mapas:

```
NamedMap<Long, List<TimestampPreference>> userPrefBuyMap =
      NamedMapFactory.getMap(RecommendConstants.USERPREF_BUY_MAP);
```

La clase *NamedMap* implementa la interfaz Map de java por lo que accedemos al IMDG como si de un map local se tratase.

3.3.2 ScaleOut hServer: Algoritmos de clasificación MapReduce

La clasificación de las preferencias se va a realizar mediante la ejecución de algoritmos *MapReduce* sobre el servidor *hServer*.

MapReduce es un modelo de programación utilizado para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de servidores. Este tipo de algoritmos se basan en dos funciones ejecutadas de forma secuencial:

- La función map() se encarga del mapeo de cada ítem de la entrada de datos. En nuestro caso se encargaría de separar, por ejemplo, aquellas preferencias que corresponden a un usuario del conjunto de todas ellas. Se encargará del acceso a *HBase* para la lectura de las preferencias almacenadas.
- La función reduce() se encarga de agrupar todas las preferencias con la misma clave (por ejemplo de un mismo usuario) generadas en el proceso de mapeo.

El lanzamiento de estos algoritmos de clasificación se realiza mediante un proceso Batch que se encarga de actualizar los datos en el IMDG con las nuevas preferencias de *HBase* insertadas desde la última ejecución.

Para realizar esta implementación se ha usado un programador de *Spring Task*. Para ello basta con definir la clase que implementa el algoritmo como un bean de *Spring*:

```
@Component
public class MapReduceJob implements InitializingBean {
```

Y programar su lanzamiento en la configuración de *Spring (applicationContext.xml):*

```
<task:scheduler id="programador" pool-size="10"/>

<task:scheduled-tasks scheduler="programador">
      <task:scheduled ref="mapReduceJob" method="execute" fixed-delay=
            "${scheduler.jobDelay}"/>
</task:scheduled-tasks>
```

Se ha extraído la variable *scheduler.jobDelay* al fichero *application.properties* para configurar el intervalo en milisegundos entre ejecuciones.

Para permitir el acceso desde *Scaleout* a Hbase se ha usado el formato TableInputFormat del API de HBase así como implementaciones de la interfaz *RowMapper* de *Spring for Apache Hadoop*.

Para la creación de la aplicación standalone se ha usado el framework *Spring Boot* generando una clase principal de la siguiente forma:

```
@SpringBootApplication
@ImportResource({"classpath*:applicationContext.xml"})
@EnableScheduling
public class Application {

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }

}
```

De esta forma conseguimos ejecutar la aplicación dentro de un contexto *Spring*.

Por último se ha configurado el plugin de Maven *Apache Maven Assembly Plugin* para que nos permita empaquetar la aplicación y todas sus dependencias como un zip ejecutable sobre cualquier entorno. Para ello una vez configurado ejecutamos en el directorio raíz del proyecto:

```
mvn clean package
```

Si todo ha ido bien nos dejará en el directorio *target* del proyecto *recommend-scaleout* un fichero zip autocontenido. Lo descomprimimos:

```
unzip recommend-scaleout-0.0.1-SNAPSHOT-application.zip
```

Y dentro de la carpeta generada ejecutamos:

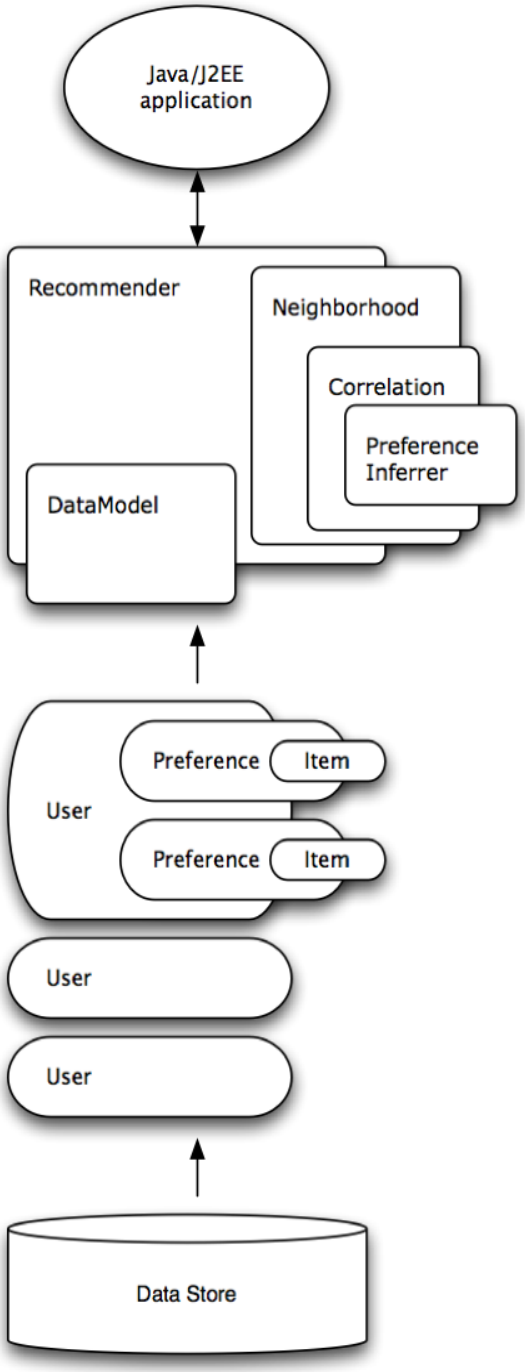
```
./start.sh
```

Vemos en el log de pantalla el resultado de la ejecución constante del algoritmo. Podemos configurar la salida del log en el fichero *conf/log4j.properties*.

3.4 Mahout

Mahout es un proyecto de la Apache Software Foundation que tiene como finalidad construir un entorno sobre el que crear algoritmos escalables de *Machine Learning*. Su enfoque principal consiste en las áreas de filtrado colaborativo, clustering y clasificación. Muchas de las implementaciones incluidas usan *Apache Hadoop*. *Mahout* proporciona además librerías Java para las operaciones matemáticas más comunes. *Mahout* está actualmente en fase de desarrollo y el número de algoritmos implementados crece continuamente.

Nuestro objetivo es integrar esta librería con *ScaleOut* para aprovechar la gran base de algoritmos ya implementados en nuestro recomendador. A continuación vamos a estudiar los componentes principales de *Mahout* para ver la forma en que se puede realizar esta integración.



3.4.1 Recommender

Un *Recommender* es la abstracción base de *Mahout*. Dado un *DataModel,* este componente puede producir recomendaciones. Las implementaciones más comunes incluidas en *Mahout* son el *GenericUserBasedRecommender* o el *GenericItemBasedRecommender*.

3.4.2 UserNeighborhood

En un motor de recomendaciones basado en usuarios, las recomendaciones se generan encontrando una cercanía o "neighborhood" de usuarios similares a uno dado. El objeto *UserNeighborhood* define la forma de medir esta cercanía, normalmente basándose en un objeto *UserSimilarity*.

3.4.3 UserSimilarity

Este objeto define la noción de similitud entre dos usuarios. Dados dos usuarios nos devuelve un valor indicando la similitud entre ellos. Es una parte crucial de un motor de recomendaciones. El objeto *ItemSimilarity* está también disponible para definir similitud entre ítems.

3.4.4 DataModel

Un *DataModel* proporciona a un *Recommender* la interfaz a través de la cual acceder los datos de preferencias de los usuarios. Los usuarios y los ítems son identificados por un valor ID dentro del framework. Además este ID debe ser de tipo *Long*. Los objetos *Preference* o *PreferenceArray* encapsulan la relación de preferencia entre un usuario y un ítem.

Mahout incluye implementaciones para acceder a datos almacenados en un fichero (*FileDataModel*) o en una base de datos MySQL (*MySQLJDBCDataModel*).

3.4.5 Integración con ScaleOut: La interfaz DataModel

La interfaz *org.apache.mahout.cf.taste.model.DataModel* representa un repositorio de información que será usado por los algoritmos de *Mahout* para obtener información sobre las preferencias de los usuarios. Dado que disponemos del servidor *ScaleOut* que nos proporciona la potencia del *DataGrid* en memoria, debemos conseguir que *Mahout* acceda directamente a estos datos en memoria y esta interfaz es un buen punto de entrada para ello.

El objeto más importante con el que trabaja el *DataModel* es el objeto del tipo *Preference*. Este objeto encapsula la información de la preferencia de un usuario por un ítem. Debe proporcionar la siguiente información:

- getUserID(): Devuelve un long con el identificador del usuario
- getItemID(): Devuelve un long con el identificador del ítem.
- getValue(): Devuelve un float con el valor de la preferencia del usuario por el ítem.

Las operaciones más importantes que define esta interfaz son:

- getUserIDs(): Devuelve un iterador para recorrer los identificadores de los usuarios en orden.
- getPreferencesFromUser(long userID): Devuelve un array con los objetos Preference asociados al usuario.
- getItemIDsFromUser(long userID): Devuelve el conjunto de identificadores de items que están asociados al usuario.

- getItemIDs(): Devuelve un iterador para recorrer los identificadores de los items en orden.
- getPreferencesForItem(long itemID): Devuelve un array con los objetos *Preference* asociados a un determinado ítem.
- getPreferenceValue(long userID, long itemID): Devuelve el valor de la preferencia de un usuario por un ítem.

Estas son las operaciones que se han implementado para permitir a *Mahout* acceder directamente a los datos almacenados en el *DataGrid* de *ScaleOut*.

3.4.6 Modelo almacenado en el IMDG

Debemos definir el modelo de los datos que almacenaremos en el *DataGrid*. La idea es almacenar la información de forma que facilite el trabajo del recomendador. Como hemos visto, la clasificación de la información se realizará en segundo plano mediante trabajos *MapReduce*.

Observando las operaciones que debemos implementar vemos que vamos a necesitar por una parte la lista de ítems por los que tiene preferencia un usuario y por otra la lista de usuarios que tienen preferencia por cada ítem. Vamos a realizar esta clasificación de forma que las operaciones implementadas por el *DataModel* sean casi directas.

La información de la que partimos en la tabla de *HBase* tiene la siguiente estructura:

UserId	ItemId
1	1
1	2
2	1
2	3
3	2
3	3

Mediante un primer trabajo *MapReduce* realizaremos la siguiente clasificación de la información:

UserId	ItemIds
1	{1, 2}
2	{1, 3}
3	{2, 3}

Y mediante un segundo *MapReduce* obtenemos otra tabla con la lista de usuarios que tienen preferencia por cada ítem:

ItemId	UserIds
1	{1, 2}
2	{1, 3}
3	{2, 3}

Además almacenaremos estas listas ordenadas de forma que ahorraremos este trabajo al *DataModel*.

Para obtener estos resultados se programan dos trabajos *MapReduce* que se ejecutan sobre el servidor de *ScaleOut* de forma paralela con lo que nos aseguramos un buen rendimiento.

La principal ventaja de esta aproximación es el consumo de CPU. Aunque el proceso de clasificación de la información es bastante costoso, al realizarse en segundo plano no va a impactar en los resultados. Por contra, la entrega de datos a *Mahout* va a tener un coste prácticamente independiente del número de interacciones una vez clasificada la información.

3.5 Cliente de prueba (recommend-clientapp)

Se ha implementado una aplicación web a modo de ejemplo de uso de la interfaz REST.

Dado que los datos que tenemos cargados en el motor de recomendaciones consisten en un conjunto de 100 ítems, hemos descargado (http://www.flaticon.com/packs/logistics-delivery) un conjunto de 100 imágenes que van a representar cada uno de estos ítems.

El funcionamiento de esta aplicación consistirá en mostrar un conjunto de elementos y al pulsar sobre cada uno de ellos se sustituirán los ítems mostrados por los ítems recibidos como recomendación del ítem pulsado. Además como añadido mostramos una caja de texto en la que podremos ir cambiando el id del usuario, un combo que nos permitirá seleccionar la operación deseada y un área de texto en el que se mostrará el log de operación.

3.5.1 Implementación

La aplicación consiste en un único fichero html llamado *inicio.html* que carga tanto el script de la aplicación REST como un script propio:

```
<script type="text/javascript"
      src="http://192.168.2.160:8080/recommend-web/js/recommend.js"></script>
<script type="text/javascript" src="js/client.js"></script>
```

Es importante cargar en primer lugar el javascript del servidor REST ya que en el javascript local tenemos que sobrescribir las funciones de procesamiento de los datos devueltos. En nuestro caso únicamente vamos a usar la función *addAndGetRecommendations* y por lo tanto solo implementaremos su procesamiento asociado:

```
Recommender.processAddAndGetRecommendations = function(data) {
      var div = document.getElementById("producto");
      while (div.hasChildNodes()) {
            div.removeChild(div.firstChild);
      }
```

```
	var logTxt = "Recomendaciones: ";
	for (var i = 0; i < data.length; i++) {

		logTxt += "[itemID: " + data[i].itemID + " value: " + data[i].value +
"]";

		var img = document.createElement("img");
		img.setAttribute("src", "img/" + images[data[i].itemID]);
		img.setAttribute("data-id", data[i].itemID);

		img.onclick = function() {
			getRecommendation(this.getAttribute("data-id"));
		};

		div.appendChild(img);
	}

	var log = document.getElementById("proceso");
	log.innerHTML = logTxt + "<br />" + log.innerHTML;
};
```

Vamos a asociar la llamada a la función REST con el evento click sobre la imagen inicial. Declaramos la función:

```
function getRecommendation(itemID) {
	var userID = document.getElementById("usuario").value;
	var op = document.getElementById("operacion").value;

	var logTxt = "Solicitando recomendaciones [userID: " + userID + " itemID: " +
itemID + " operacion: " + op + "]";

	var log = document.getElementById("proceso");
	log.innerHTML = logTxt + "<br />" + log.innerHTML;

	Recommender.addAndGetRecommendations(userID, itemID, op);
};
```

Y por último en la imagen:

```
<img src="img/2440.png" onclick="getRecommendation(1);"/>
```

De esta forma al ir pulsando sobre las imágenes se realizará la llamada al recomendador sustituyendo las imágenes por las imágenes de los productos recomendados.

4 PRESTACIONES

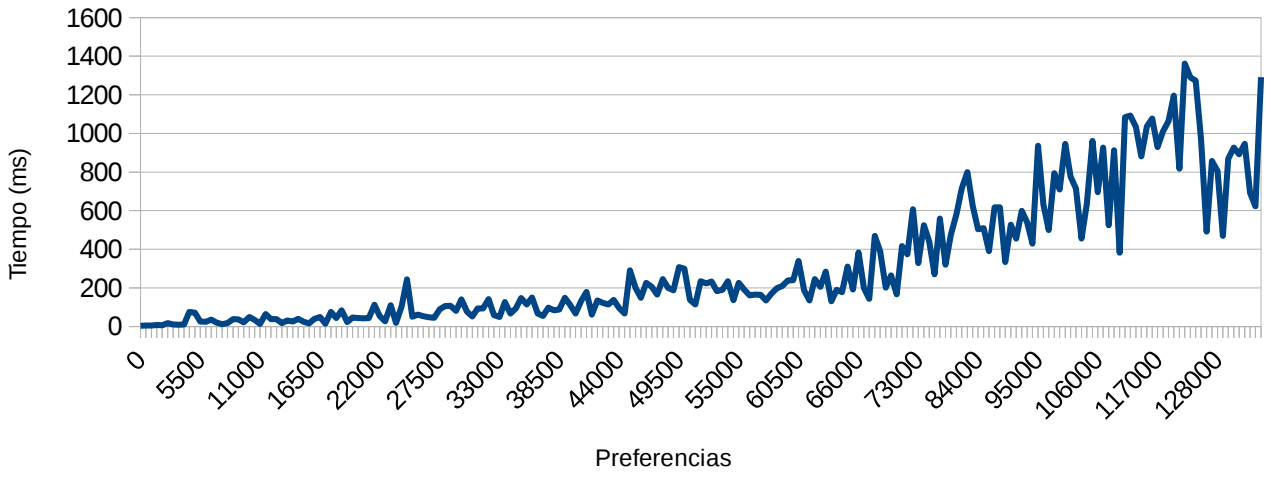
En este apartado vamos a evaluar el rendimiento del sistema de recomendaciones. El sistema completo se ha implementado dentro de una máquina virtual VirtualBox con las siguientes características:

- Hardware: AMD Phenom 9750 Quad Core (4 x 2.4GHz), asignadas 6GB de memoria DDR2 800MHz
- Sistema Operativo: LMDE (Linux Mint Debian Edition) con el entorno de escritorio basado en XFCE y los últimos parches disponibles a la fecha de la prueba.
- Al tratarse la mayor parte del sistema de software libre, en la medida de lo posible se han utilizado las últimas versiones disponibles de todo el software y librerías.

El método de prueba ha consistido en:

- Por una parte lanzamos el proceso Batch que programa la ejecución del algoritmo *MapReduce* de forma que lance los trabajos cada minuto.
- Lanzamos un proceso que inserte peticiones aleatorias de forma constante.
- Cada 500 inserciones pedimos una recomendación al motor y calculamos el tiempo que tarda en devolverla.

Dejamos todo el sistema funcionando durante varias horas y generamos la siguiente gráfica con los resultados:



A la vista de los resultados se pueden hacer los siguientes comentarios:

- En primer lugar la prueba se detuvo cuando el motor tenía unas 150000 preferencias insertadas. En este punto el sistema operativo comenzó a paginar memoria en disco y a responder de forma muy lenta.
- El uso de memoria del proceso de *ScaleOut* ha sido prácticamente constante (unos 2Gb de memoria) durante toda la prueba. Por lo visto en la documentación, el IMDG reserva una cierta cantidad de memoria en función de la RAM libre en la máquina para su funcionamiento.
- Es importante destacar que se ha ejecutado toda la prueba sobre una máquina virtual que, a priori, va a tener un rendimiento inferior a la ejecución sobre una máquina física. Además en esta prueba se ha ejecutado sobre la misma máquina varios servicios que en un entorno real pueden

distribuirse en distintas máquinas (servidor de BD, clúster de servidores de *ScaleOut*, proceso de *MapReduce*, proceso de carga y petición de recomendaciones).

En resumen, el resultado de la prueba ha sido satisfactorio. El tiempo de respuesta permanece aceptable y más o menos constante hasta cierto número de preferencias en el que empieza a aumentar de forma no lineal. Parece bastante probable que este número a partir de el cual el rendimiento empieza a degradarse dependa de la cantidad de memoria disponible en el clúster, con lo cual en un entorno real de producción este número será mucho más elevado que en esta prueba, no obstante, parece interesante incorporar algún mecanismo de limpieza de preferencias antiguas.

5 CONCLUSIONES

En este proyecto se ha conseguido integrar varias tecnologías para obtener un motor de recomendaciones robusto y eficiente. La parte más importante ha sido la integración entre la librería *Apache Mahout* y los productos de *ScaleOut*. Esta integración va a permitir a nuestro recomendador el aprovechar toda la base de código ya implementada por la comunidad de *Mahout* y además aprovechar la potencia del IMDG.

El objetivo del proyecto se ha centrado en crear un piloto de un recomendador que funcione. En este proceso se han dejado de lado algunos aspectos que sería recomendable tener en cuenta antes de llegar a usarse en entornos de reales de producción.

En primer lugar sería interesante montar un clúster real de servidores *ScaleOut* y realizar pruebas de rendimiento midiendo hasta que nivel de carga se obtienen respuestas en un tiempo aceptable.

En cualquier caso parece necesario implementar algún mecanismo de limpieza de preferencias antiguas cuando se llegue a un punto de alta carga. Las preferencias antiguas influyen menos que las más actuales en el resultado final de las recomendaciones por lo que si se establece un límite de limpieza razonable, esta operación no va a empeorar la calidad de las recomendaciones.

Para ofrecer recomendaciones como SaaS también puede ser útil incluir algún tipo de gestión de tiendas con acceso al sistema, gestión de sus permisos, etc. Además puede ser interesante ofrecer a las tiendas un portal al que puedan acceder para gestionar sus datos, por ejemplo dar más peso a las recomendaciones sobre algún producto que esté en promoción o cosas por el estilo.

Por último en la implementación realizada, aunque los datos se encuentran en el IMDG, el proceso recomendador de *Mahout* se ejecuta dentro de los servidores que ejecutan la aplicación web. Sería interesante de cara al rendimiento estudiar la posibilidad de ejecutar estos procesos dentro de la infraestructura de *ScaleOut*, en concreto habría que estudiar la adaptación de la librería de *Mahout* para ejecutarse dentro de *ScaleOut Compute Server*.

6 REFERENCIAS

Inteligencia Colectiva: https://es.wikipedia.org/wiki/Inteligencia_colectiva
Documento sobre RecommendInCloud de César Carralero: http://qualityobjects.com/images/data/White%20Paper%20RecommendInCloud.pdf
Hbase: https://hbase.apache.org/
ScaleOut: http://www.scaleoutsoftware.com/
Apache Hadoop: https://hadoop.apache.org/
Apache Mahout: https://mahout.apache.org/