Ejercicios de Map Reduce

Pensando en paralelo

Universitat Oberta de Catalunya

Cuando trabajamos con entornos de almacenamiento de datos distribuidos, procesar los datos de forma centralizada puede tener unos costes de computación y transmisión de datos inasumibles. En estos casos es conveniente utilizar técnicas de procesamiento distribuido, como por ejemplo *map reduce*.

En este documento se describe brevemente el proceso de map reduce, sus principales fases y se proponen algunos ejercicios resueltos en detalle para facilitar su comprensión.

Autores

Antonio Sarasa Cabezuelo

Joan Anton Pérez Braña

Jordi Conesa i Caralt (revisor)

Índice

Histórico de revisiones	
Versión 1.0	Marzo 2020

Pensando en paralelo

Supongamos que disponemos de un conjunto de datos global de reservas de vuelo y deseamos realizar algo tan simple como contar el número de vuelos de cada aerolínea. Éste será nuestro escenario de ejemplo. Para un programa que se ejecute centralizado, ésta es una operación simple.

El pseudocódigo que se muestra a continuación es bastante sencillo: configura un array para almacenar la cantidad de veces que se encuentra con cada aerolínea, y luego, a medida que se lea secuencialmente cada registro de origen, incrementar el contador de la aerolínea correspondiente.

```
Crear una matriz bidimensional.

Crear una fila para cada aerolínea de forma que:
      Se registra en la primera columna los distintos valores
      correspondientes al código de la aerolínea.
      Se registra en la segunda columna el entero cero.

Para cada línea de datos de reservas de vuelo:
      Leer el código de la aerolínea.
      Encontrar la fila en la matriz que coincide con el código de
      la aerolínea.
      Incrementar el contador en la segunda columna en uno.

Finalmente, mostrar para cada fila de la matriz bidimensional, el
código de la aerolínea de la primera columna y los totales
obtenidos en la segunda columna.
```

La cuestión es que no se podrá utilizar este código y ejecutarlo con éxito cuando los datos de las distintas reservas de vuelo están almacenados en un sistema distribuido. Aunque éste es un ejemplo simple, es necesario pensar en paralelo mientras se codifica la correspondiente aplicación.

En esencia, MapReduce es un modelo de programación para procesar conjuntos de datos que se almacenan de forma distribuida en distintos nodos de un clúster.

La información, ubicada en cada nodo del sistema, se procesará en paralelo con el mismo algoritmo.

Las aplicaciones MapReduce están formadas por 3 fases:

- **Fase Map**: se realiza un procesamiento sobre los conjuntos de datos de interés para generar datos en forma de parejas (clave,valor). Esta fase se ejecuta localmente en los nodos donde se almacenan los datos.

- **Fase Shuffle and Sort**: a partir de las parejas (clave/valor) generadas en la fase de map, se crean nuevas parejas (clave,valor) que agrupan los valores asociados a una misma clave. Así, a partir de cada clave, se crea una nueva pareja con el valor de la clave y un vector que contiene los distintos valores asociados a la clave.

- **Fase reduce**: se recuperan los pares (clave,valor) generados en la fase anterior y se procesan todos los datos asociados a una misma clave a la vez.

Aunque éste es un ejemplo simple, el desarrollo de la versión paralela requiere un enfoque completamente diferente. El siguiente pseudocódigo muestra cómo calcular en paralelo el número de reservas de vuelo para cada aerolínea.

Map

```
Para cada línea de datos de vuelo
     Leer los datos actuales y extraer el código de la aerolínea.

     Generar una pareja (clave, valor) formada por el código de
     la aerolínea como clave y el número uno como valor.
```

Shuffle & Sort

```
Recuperar la lista de parejas clave/valor de la fase map.

Ordenar los datos por clave.

Se generan nuevas parejas (clave/ valor) agrupando los valores de
las parejas que tienen un mismo código de aerolínea. La nueva
clave será el código de aerolínea y el valor un vector de unos
(con una posición por cada ocurrencia (clave/valor) procesada).
```

Reduce

```
Leer la lista de parejas claves y vectores de valores de la fase
Shuffle and Sort para cada código de aerolínea.

Generar un nuevo par (clave, valor) con el código de aerolínea
como clave, y tamaño (o número de unos)del vector asociado como
valor.
```

```
Mostrar la nueva lista de pares (clave, valor)
```

El código muestra cómo abordar el mismo problema desde una perspectiva de procesamiento de datos en paralelo. Como necesitamos totales, tuvimos que dividir esta aplicación en fases.

1. La primera fase es la fase de mapeo (*map*), que es donde cada conjunto de datos referido a un vuelo se procesa individualmente. Aquí, extraemos el código de la aerolínea del vuelo y generamos una pareja clave/valor, con el código de la aerolínea como clave y un número “1” como valor. Esta operación se ejecuta contra cada vuelo del conjunto de datos.
2. Después de procesar cada vuelo de forma local, debe asegurarse de que los valores generados (los enteros con valor “1”) se agrupen para cada clave, que es el código de la aerolínea, y luego se ordenan por clave. Esto se conoce como la fase de barajar y ordenar (*shuffle & sort*).
3. Finalmente (*reduce*), se suma el número total de “1” de unidades para cada aerolínea, lo que le da el total de reservas de vuelo para cada aerolínea.

Como se puede ver, hay poco en común entre la versión en serie del código y la versión paralela. A continuación, proponemos algunos ejercicios para aprender de forma práctica a trabajar con una mentalidad en paralelo.

Ejercicios

A continuación proponemos algunos ejercicios para practicar el diseño de agregados. Las soluciones detalladas a los mismos se encuentran al final de este documento.

Ejercicio 1

Considerar dos tablas de una base de datos relacional que almacenan información sobre los datos personales de los estudiantes y sobre las asignaturas en las que se encuentran matriculados. En este sentido, los campos de la tabla `Estudiantes` son: `dni, nombre, apellido1, apellido2, email, ciudad`. Todos los campos son cadenas. Y los campos de la tabla `Matricula` son: `dni, asignatura, cuatrimestre, créditos, carácter`. Todos los campos son cadenas salvo el campo `créditos` que es número real de un decimal.

Suponiendo que se tuviera dos listas que contuvieran las tuplas para cada tabla (Estudiantes y Matricula), se pide simular utilizando Map-Reduce la siguiente consulta SQL:

```
SELECT nombre, apellido1, apellido2, asignatura
FROM Estudiantes, Matricula
WHERE Estudiantes.dni=Matricula.dni AND
      ciudad=”Zaragoza” AND
      créditos>3.5 AND
      carácter=”Obligatoria”;
```

Describid los cálculos realizados por las funciones de Map y Reduce, usando un ejemplo para ilustrarlo.

Ejercicio 2

Considerar un fragmento de la base de datos relacional de una librería que almacena información sobre los libros vendidos. Los campos de la tabla Ventas son: isbn, título, escritor, editorial, ejemplares, y ganancias. Todos los campos son cadenas salvo los campos ejemplares y ganancias que son enteros positivos.

Suponiendo que se tuviera una lista que contuviera las tuplas de la tabla descrita, se pide simular utilizando Map-Reduce la siguiente consulta SQL:

```
SELECT escritor, editorial, SUM(ejemplares), SUM(ganancias)
FROM ventas
GROUP BY escritor, editorial
```

Describid los cálculos realizados por las funciones de Map y Reduce, usando el siguiente ejemplo que se muestra en formato tabular:

ISBN	TÍTULO	ESCRITOR	EDITORIAL	EJEMPLARES	GANANCIAS
234534J	El Quijote	Miguel de Cervantes	Anaya	45	546
134444H	Novelas ejemplares	Miguel de Cervantes	Anaya	34	234
244553I	Noche de reyes	William Shakespeare	Santillana	23	256
267787T	El mercader de Venecia	William Shakespeare	Alianza	15	345
234567R	Drácula	Bram Stocker	Alianza	34	456
545545E	El Lazarillo de Tormes	Anónimo	Cátedra	35	245
987685E	El conde Lucanor	Don Juan Manuel	Cátedra	27	346

Soluciones

A continuación se encuentran los soluciones a los ejercicios propuestos.

Ejercicio 1

Para explicar la solución se va a considerar el siguiente ejemplo, que representa la lista de las tuplas de la tablas:

```
Estudiantes=[(“5456644E”,”Teresa”, “Sanz”, “López”,”psanzl@gmail.com” ,”Zaragoza”),
        (“4566333R”,”Isabel”, “Esquinas”, “Mol”,”isaesmol@gmail.com” ,”Barcelona”),
        (“5134553E”,”Mario”, “García”, “Martín”,”magamar@hotmail.com” ,”Valencia”),
        (“3456733H”,”Antonio”, “Perea”, “Libertad”,”anpeli@yahoo.com” ,”Madrid”),
        (“34566343D”,”Mamen”, “Mérida”, “Salcedo”,”mamesa@yahoo.com” ,”Zaragoza”),
        (“43435553J”,”Eva”, “Sol”, “Tequila”,”esolte@hotmail.com” ,”Pamplona”),
        (“3456453R”,”Juán”, “Marcos”, “Pérez”,”jmperezl@gmail.com” ,”Zaragoza”)]

Matricula=[(“5456644E”,”Matemática Discreta”, “Primero”,4.0, “Obligatoria”),
        (“4566333R”,”Historia del arte”, “Primero”,5.5, “Obligatoria”),
        (“5456644E”,”Teoría de la computabilidad”, “Segundo”,3.5, “Optativa”),
        (“43435553J”,”Álgebra”, “Segundo”,8.0, “Obligatoria”),
        (“4566333R”,”Arte Medieval”, “Primero”,4.5, “Obligatoria”),
        (“3456733H”,”Química I”, “Segundo”,6.5, “Optativa”),
        (“34566343D”,”Derecho procesal”, “Segundo”,3.0, “Obligatoria”),
        (“43435553J”,”Enfermedades infecciosas”, “Primero”,3.5, “Obligatoria”),
        (“3456453R”,”Ingeniería del Software”, “Segundo”,4.5, “Obligatoria”)]
```

Operación de *Map*:

La operación *Map* deberá procesar las dos listas que tomará como entrada. El resultado de procesar una tupla de las listas será que la función *Map* devuelve tuplas clave-valor, con el campo ***dni*** como clave y la tupla como el valor:

```
función MAP(key, value)
        /* key: nombre documento
         value: contenido del documento */
        para cada tupla t ∈ value hacer:
                emitir (t.DNI , t);
fin función
```

El resultado de la salida de la función sería el siguiente:

```
("5456644E",("5456644E","Teresa", "Sanz", "López","psanzl@gmail.com" ,"Zaragoza")),
("4566333R", ("4566333R","Isabel", "Esquinas", "Mol","isaesmol@gmail.com" ,"Barcelona")),
("5134553E" ,("5134553E","Mario", "García", "Martín","magamar@hotmail.com" ,"Valencia")),
("3456733H" ,("3456733H","Antonio", "Perea", "Libertad","anpeli@yahoo.com" ,"Madrid")),
("34566343D",("34566343D","Mamen", "Mérida", "Salcedo","mamesa@yahoo.com" ,"Zaragoza")),
("43435553J", ("43435553J","Eva", "Sol", "Tequila","esolte@hotmail.com" ,"Pamplona")),
("3456453R", ("3456453R","Juán", "Marcos", "Pérez","jmperezl@gmail.com" ,"Zaragoza")),
("5456644E",("5456644E","Matemática Discreta", "Primero",4.0, "Obligatoria")),
("4566333R"),("4566333R","Historia del arte", "Primero",5.5, "Obligatoria")),
("5456644E", ("5456644E","Teoría de la computabilidad", "Segundo",3.5, "Optativa")),
("5134553E", ("5134553E","Álgebra", "Segundo",8.0, "Obligatoria")),
("4566333R", ("4566333R","Arte Medieval", "Primero",4.5, "Obligatoria")),
("3456733H"," ("3456733H","Química I", "Segundo",6.5, "Optativa")),
("34566343D"," ("34566343D","Derecho procesal", "Segundo",3.0, "Obligatoria")),
("43435553J", ("43435553J","Enfermedades infecciosas", "Primero",3.5, "Obligatoria")),
("3456453R", ("3456453R","Ingeniería del Software", "Segundo",4.5, "Obligatoria"))
```

Antes de pasar la lista de tuplas a la operación *Reduce*, se agrupan los valores por clave (función Shuffle) y se obtiene:

```
("5456644E",[("5456644E","Teresa",  "Sanz",  "López","psanzl@gmail.com"  ,"Zaragoza"),
("5456644E","Matemática Discreta", "Primero",4.0, "Obligatoria")), ("5456644E","Teoría de
la computabilidad", "Segundo",3.5, "Optativa")]),
("4566333R",[ ("4566333R","Isabel", "Esquinas", "Mol","isaesmol@gmail.com" ,"Barcelona"),
("4566333R","Historia del arte", "Primero",5.5, "Obligatoria")]),
("5134553E",[ ("5134553E","Mario", "García", "Martín","magamar@hotmail.com" ,"Valencia"),
("5134553E","Álgebra", "Segundo",8.0, "Obligatoria")]),
("3456733H",[ ("3456733H","Antonio", "Perea", "Libertad","anpeli@yahoo.com" ,"Madrid"),
("3456733H","Química I", "Segundo",6.5, "Optativa")]),
("34566343D",[ ("34566343D","Mamen", "Mérida", "Salcedo","mamesa@yahoo.com" ,"Zaragoza"),
("34566343D","Derecho procesal", "Segundo",3.0, "Obligatoria")]),
("43435553J",[ ("43435553J","Eva", "Sol", "Tequila","esolte@hotmail.com" ,"Pamplona"),
("43435553J","Enfermedades infecciosas", "Primero",3.5, "Obligatoria")])
("3456453R",[ ("3456453R","Juán", "Marcos", "Pérez","jmperezl@gmail.com" ,"Zaragoza"),
("3456453R", "Ingeniería del Software", "Segundo",4.5, "Obligatoria")])
```

Operación *Reduce*:

Recibe como argumentos una clave formada por el ***dni*** y una lista de las tuplas que comparten como valor del campo dni, el valor de la clave. En la operación *Reduce* se filtran las tuplas que se reciben de acuerdo con las condiciones dadas. Se va a ilustrar en pseudocódigo como sería la lógica de la función reduce:

```
función reduce (key, values)
        encontrado:=falso
         i:=1
         resultados:=[]
         mientras i<=longitud(values) hacer
            /*Si la longitud del elemento es 6 corresponde a una tupla de la tabla
              Estudiantes y solo nos interesan aquellas tuplas donde la ciudad es
              Zaragoza */
               si longitud(values[i])=6 y
           values[i][6]=”Zaragoza” entonces
                        nombre:=i[2]
                        apellido1:=i[3]
                        apellido2:=i[4]
                       encontrado:=cierto
                si no entonces
                   /* Sino, es la tabla matricula, y solo nos interesa las asignaturas si
                       tienen más de 3.5 créditos, son obligatorias y el estudiante es
                       de Zaragoza */
             si longitud(values[i])=5 y
                        values[i][4]>3.5 y
                         values[i][5]=”Obligatoria” y
      encontrado entonces
                              insertar(resultados,nombre,apellido1,apellido2,values[i][2])
                     fin si
                fin si
                i=i+1
         fin mientras
         si longitud(resultados)>0 entonces
            para cada pos en resultados hacer
               imprimir(resultados[pos])
           fin para
         fin si
fin función
```

Con los valores anteriores, si se aplica la función se obtendría el siguiente resultado:

```
("Teresa", "Sanz", "López", "Matemática Discreta"),
("Juan", "Marcos", "Pérez", "Ingeniería del Software")
```

Ejercicio 2

Operación *Map*

La operación Map deberá procesar la lista que tomará como entrada. El resultado de procesar una tupla de la lista será que la función Map devuelve una tupla clave-valor donde la clave estará formada por la tupla correspondiente al escritor y a la editorial, y el valor será otra tupla que contendrá los ejemplares y las ganancias:

```
función MAP(key, value)
        /* key: nombre documento
         value: contenido del documento */

        para cada tupla t ∈ value hacer:
                emitir({escritor:t.escritor,editorial:t.editorial} ,
                        {ejemplares: t.ejemplares, ganancias: t.ganancias});
fin función
```

El resultado de su ejecución sería el siguiente:

```
(("Miguel de Cervantes","Anaya"),(45,546)),
(("Miguel de Cervantes","Anaya"),(34,234)),
(("William Shakespeare","Santillana"),(23,256)),
(("William Shakespeare","Alianza"),(15,345)),
(("Bram Stocker","Alianza"),(34,456)),
(("Anónimo","Cátedra"),(35,245)),
(("Don Juan Manuel","Cátedra"), (27,346))
```

Antes de pasar la lista de tuplas a la operación *Reduce*, se agrupan los valores por clave (función *Shuffle*) y se obtiene:

```
(("Miguel de Cervantes","Anaya"),[(45,546), (34,234)],
(("William Shakespeare","Santillana"),[(23,256)]),
(("William Shakespeare","Alianza"),[(15,345)]),
(("Bram Stocker","Alianza"),[(34,456)]),
(("Anónimo","Cátedra"),[(35,245)]),
(("Don Juan Manuel","Cátedra"), [(27,346)])
```

Operación *Reduce*:

Recibe como argumentos una clave que representa un par formado el escritor y la editorial y una lista de las tuplas ejemplares vendidos y ganancias obtenidas. Así en la operación *Reduce* se suman los valores que contiene la lista y se emite la tupla formada por la clave y la suma de los ejemplares y la suma de las ganancias.

```
función REDUCE (key, valueList)
        /* key: par escritor,editorial
         valueList: lista de pares ganancias,ejemplares */

        // inicializar resultado
        res := { ejemplares: 0, ganancias: 0};

        para cada par v ∈ valueList hacer:
                res.ejemplares += v.ejemplares;
                res.ganancias  += v.ganancias;
        fin para
        emitir({escritor:key.escritor,editorial:key.editorial} ,
                {SumaEjemplares:res.ejemplares,SumaGanancias:res.ganancias
                });
fin función
```

Con los valores anteriores, se obtendría los siguientes resultados:

```
("Miguel de Cervantes","Anaya",79, 780),
("William Shakespeare","Santillana",23,256),
("William Shakespeare","Alianza",15,345),
("Bram Stocker","Alianza",34,456),
("Anónimo","Cátedra",35,245),
("Don Juan Manuel","Cátedra", 27,346)
```