**MapReduce, algunos apuntes**

Muchos problemas pueden resolverse mediante el modelo de map-reduce, pero también hay muchos que no. Por eso conviene centrarse un poco en las ideas conceptuales del esquema de programación de MapReduce.

Este modelo procede de la programación funcional y consiste en resolver un problema mediante dos operaciones:

* Map, que aplica una misma función a una lista de elementos
* Reduce, que combina la lista de los resultados producidos por map.

Veamos dos ejemplos sencillos con los que introducir los conceptos básicos uno a uno:

* El del sumatorio:

Una observación crucial es que la función map se puede aplicar a los elementos de la lista en cualquier orden, y por tanto, también en paralelo. A la función que se aplica a los elementos se le suele llamar mapper, función mapeadora. En nuestro caso, la función mapeadora calcula el elemento k-ésimo de la secuencia, para cada k. En nuestro ejemplo, la función mapeadora es la siguiente:

La lista de términos es la siguiente:

términos = map( , range(1, n+1))

La función reductora, el reducer, ha de ser conmutativa y asociativa. De este modo, los pares de resultados que da la función map se pueden combinar en cualquier orden, y en particular en paralelo y a medida que vayan siendo generados por los mappers. En nuestro caso, es la siguiente:

Es decir, se trata de una simple suma. Por lo tanto,

sumatorio = reduce( , términos).

En realidad, la función mapper en el modelo MapReduce hace algo más que procesar elemento a elemento: esta función no calcula simplemente un valor, sino una colección de pares clave-valor.

* Queremos ahora contabilizar cuántas veces aparece el carácter “a”, cuántas el carácter “b”, etc., hay en un texto formado por una colección de líneas.

|  |
| --- |
| calabaza  lima |

El mapper toma cada elemento (una línea) de la lista de entrada y devuelve una lista de pares, clave-valor en un dominio diferente:

(Te preguntarás por qué no hemos hecho una función más útil, como la siguiente:

Pero déjame que conteste a esto más tarde.)

El reducer toma todos los valores de cada clave y los combina:

Para que todo esto funcione, basta con definir el mapper, que actúa sobre una línea así:

|  |
| --- |
| for car in línea:  yield car, 1 |

Y el reducer, que actúa sobre una lista de números así:

sum(lista)

Completando el paréntesis de antes, vemos que una función que suma las repeticiones de cada carácter, simplemente adelanta una parte del trabajo del reducer, pero no altera el resultado, y es más sencillo para nosotros dar la versión sencilla de una lista de unos.

…………………………………………………………………………………………………………………………..

El modelo de map-reduce está pensado para que una tarea se procese en un clúster formado por distintas máquinas. Desde el punto de vista práctico, la cosa funciona como sigue:

* Los datos de entrada proceden por regla general de un archivo, normalmente de gran tamaño, y se dividen en un conjunto de *particiones* de entrada, generalmente de entre 16 y 64 Mb, y cada una de estas particiones se va enviando a una máquina del clúster que esté en reposo.
* Cada máquina recibe su bloque de información y los procesa independiente de las demás, aplicando el mapper.
* El mapper va generando resultados tan pronto como los vaya calculando: justamente para eso usamos la instrucción yield, que es parecida a return, pero que no interrumpe la función, sino que sigue generando resultados.
* Cada resultado tiene la forma clave-valor, como se ha explicado. Los pares clave y valor producidos por las distintas máquinas se van agrupando en regiones y enviados nuevamente a máquinas con la finalidad de aplicarles las tereas de reductoras.
* Estas reducciones se producen en un orden indeterminado, a medida que se van generando por los mappers, agrupando (automáticamente en listas) y recibiendo por los reducers.
* Las claves juegan un papel esencial, porque frecuentemente muchas claves permiten agrupar los resultados de los mappers para su procesamiento; el ejemplo segundo (pares de nombres propios) evidencia este hecho.