MapReduce

Ejercicio N°3

Objetivos	• Diseño y construcción de sistemas orientados a objetos • Uso de buenas prácticas de programación en C++
Instancias de Entrega	**Entrega 1:** clase 8 (26/4/2016). **Entrega 2:** clase 10 (10/5/2016).
Temas de Repaso	• Definición de clases en C++ • Contenedores de STL • Excepciones / RAII
Criterios de Evaluación	• Criterios de ejercicios anteriores • Ausencia de funciones globales (salvo función 'main') • Orientación a objetos del sistema • Empleo de estructuras comunes C++ (string, fstreams, etc) en reemplazo de su contrapartida en C (char*, FILE*, etc) • Uso de **const** en la definición de métodos y parámetros • Empleo de constructores y destructores de forma simétrica • Buen uso del stack para construcción de objetos automáticos

Índice

Introducción

En enero del año 1999, Larry Page y Sergey Brin patentan un algoritmo que revoluciona la historia de la informática, y lo bautizan *PageRank*. Este algoritmo fue el primero que utilizó el motor de búsqueda *Google*, y consiste en sucesivas multiplicaciones de matrices muy grandes. Para realizar esas operaciones de manera eficiente, surge un esquema que las paraleliza: *MapReduce*. Este esquema consiste en separar un proceso que recibe y devuelve pares *(clave, valor)* en unas funciones *map()*, y otras *reduce()*. A grandes rasgos, estos dos tipos de funciones funcionan de la siguiente manera:

- Los *Map* reciben los pares *(clave, valor)* originales, digamos de la forma (k_m , v_m), y genera otros, de la forma (k_r , v_r). Lo que tengan estos pares generados, depende del proceso que estemos implementando. Lo fundamental es que luego todos los pares que tengan la misma clave se agrupan en una lista que va al mismo *Reduce*. Se puede decir que cada *Map* hace un *remapeo* de los datos originales.
- Cada *Reduce* recibe los pares generados por *Map* que tengan la misma clave agrupados en una lista, y les aplica su propio algoritmo para generar el resultado deseado.

Por ejemplo, si los *Map* generaron los pares (1, 2), (3, 4), (1, 53), habrá un *Reduce* que recibirá (1, [2, 53]), y otro que recibirá (3, [4]). Nótese que las claves de salida de los *Map* son las de entrada de los *Reduce*.
Además de estos dos, hay otros tipos de entidades que intervienen en el esquema de *MapReduce*, pero están fuera del alcance de este trabajo práctico.

Descripción

En este ejercicio, se deberá implementar un esquema *MapReduce* simple, en el que los *Mappers* deberán pertenecer a procesos *client*, y los *Reducers* deberán correr en hilos separados de un *server*.
Se deberá deducir, a partir de listas de temperaturas máximas de distintas ciudades del mundo, cuáles fueron las más calurosas del mes de marzo, día por día.

Entrada

El *input* del sistema lo recibirán los clientes por entrada estándar, y serán líneas texto plano que respetarán la forma:

```
NombreDeLaCiudadSinEspacios Temperatura DíaDeMarzo\n
```

Los *Mappers* de cada cliente (uno por ciudad) deben generar pares key-value convenientes para cada línea, y enviárselos al servidor para que éste corra los *Reduce* correspondientes.
Nota: Si en el servidor se necesita calcular una temperatura máxima **por día**, tal vez el día sea una buena clave de entrada para los *reduce*.

Protocolo de comunicación

Los pares key-value que generan los *Mappers* deberán ser transmitidos al servidor terminados por un salto de línea (‘\n’). Esto es, si se quiere transmitir el par (SoyUnaClave, SoyUnValor), se debe enviar la línea:

```
SoyUnaClave SoyUnValor\n
```

Cuando se reciba un fin de archivo por entrada estándar, se informará al servidor esta situación enviándole la cadena:

```
End\n
```

Luego, el cliente finalizará retornando 0 como código de salida.

Por otro lado, el servidor aceptará la conexión de clientes indefinidamente y procesará los pares key-value que envíen hasta recibir por entrada estándar el caracter **'q'**. Una vez que esto ocurre, el servidor dejará de aceptar conexiones, permitirá que todos los clientes finalicen su envío, procederá con el cálculo de mayores temperaturas e imprimirá los resultados por salida estándar. Por último, finalizará retornando 0 como código de salida.

Salida

Cada una de las distintas claves, que es recomendable que sean los días del mes de marzo, implicará un hilo *Reducer* en el servidor, que va a recibir una lista que contendrá todos los valores que generaron los *Mappers* para su clave.
El *output* del sistema lo imprimirá el servidor por salida estándar, con la información que calculada, y deberá tener la forma:

```
1: NombreDeLaCiudadMásCalurosaElPrimeroDeMarzo (Temperatura)\n
2: NombreDeLaCiudadMásCalurosaElDosDeMarzo (Temperatura)\n
…
31: NombreDeLaCiudadMásCalurosaElTreintaYUnoDeMarzo (Temperatura)\n
```

Formato de Línea de Comandos

Para correr el servidor, se le indicará el puerto en el que escuchará conexiones de los clientes:

```
./server port
```

Por ejemplo:

```
./server 7777
```

Al cliente también se le deberá indicar la dirección ip de la máquina en la cual está corriendo el proceso servidor:

```
./client ip port
```

Por ejemplo:

```
./client 127.0.0.1 7777
```

Códigos de Retorno

Tanto cliente como servidor devolverán 0 en cualquier caso.

Ejemplos de Ejecución

Supongamos que marzo tiene sólo tres días, para la claridad de los ejemplos:

Sólo una ciudad

Se lanza un proceso cliente, y se le ingresa por entrada estándar el siguiente texto:

```
BuenosAires 23 1\n
BuenosAires 23 2\n
BuenosAires 24 3\n
```

Los mappers van a recibir esta información y generarán, por ejemplo, pares del estilo:

```
1 23 BuenosAires\n
```

O bien:

```
2 23 BuenosAires\n
```

El servidor va a recibir estas líneas, y lanzará los hilos reducers. Como la ciudad es una sola, la solución será trivial, y el servidor imprimirá:

```
1: BuenosAires (23)\n
2: BuenosAires (23)\n
3: BuenosAires (24)\n
```

Dos ciudades

Además de las temperaturas del ejemplo anterior, se reciben en otro cliente las temperaturas de otra ciudad (notar que no tienen por qué estar ordenadas):

```
Parana 23 3\n
Parana 24 2\n
Parana 25 1\n
```

Ahora en cada hilo reducer, se recibirán datos de más de una ciudad, y el resultado final será otro:

```
1: Parana (25)\n
2: Parana (24)\n
3: BuenosAires (24)\n
```

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C (C99) / ISO C++98.
2. Está prohibido el uso de variables globales.
3. Una solución que no implemente un esquema MapReduce no es válida.
4. En el servidor, se debe sincronizar la recepción de datos con el lanzamiento de los procesos *Reduce*. Primero se reciben todos los datos de los clientes, y recién luego se procesan.

Referencias

[1] MapReduce: https://es.wikipedia.org/wiki/MapReduce
[2] PageRank: https://es.wikipedia.org/wiki/PageRank