Sistemas Operativos 1/2023
Laboratorio 3

Profesores:
Marcela Rivera (marcela.rivera.c@usach.cl)

I. Objetivos Generales

Este laboratorio tiene como objetivo aplicar los conceptos de creación de hebras, la comunicación entre estos a través de la memoria compartida y métodos de sincronización tales como barreras y/o exclusión mutua, mediante el soporte de un sistema operativo basado en el núcleo Linux y el lenguaje de programación C.

II. Objetivos Específicos

1. Conocer y usar las funcionalidades de `getopt()` como método de recepción de parámetros de entradas.
2. Crear hebras a través del uso de `pthread_create()`.
3. Proveer exclusión mutua a secciones críticas
4. Sincronizar hebras mediante el uso de funciones `pthread_barrier`.

III. Antecedentes

III.A. Procesamiento de datos

El procesamiento de grandes volúmenes de datos cada vez se hace más presente en la vida cotidiana y negocio de las empresas. Cuando se habla de BigData, se hace alusión justamente al trabajo de grandes conjuntos de datos o combinaciones de conjuntos de datos cuyo tamaño (volumen), complejidad (variabilidad) y velocidad de crecimiento (velocidad) dificultan su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales, tales como bases de datos relacionales y estadísticas convencionales o paquetes de visualización, dentro del tiempo necesario para que sean útiles.

Dependiendo del tamaño de un conjunto de datos, podemos enfrentarnos a mayor dificultad de procesamiento, por lo cual se hace necesario encontrar diferentes maneras de gestionar los datos y procesarlos.

III.B. MapReduce

MapReduce nace en Google, aun cuando esta compañía, tiene otros métodos para realizar cálculos que procesan gran cantidad de datos en bruto. La razón es que apesar de contar con diferentes metodologías y tener que realizar cálculos sencillos los datos de entrada son muy grandes y los cálculos se han distribuido a través de cientos o miles de máquinas con el fin de terminar en un tiempo razonable, lo cual requiere distribución de datos tarea muy importante que no puede presentar errores porque de lo contrario se puede obtener una gran cantidad de errores lo cual afectará los resultados y demandará mayor tiempo para corregirlos.

Como reacción a este problema, ingenieros de Google, diseñaron una nueva abstracción que permite expresar los simples cálculos a realizar, pero oculta los detalles de paralelización, tolerancia a fallos, la distribución de datos y balanceo de carga en una API.

El modelo funcional MapReduce permite paralelizar grandes cálculos fácilmente y usar nueva ejecución como el mecanismo principal para la tolerancia a fallos. Para conseguir esto, MapReduce se descompone en tres principales partes: coordinador encargado de crear las unidades de trabajo que harán los cálculos, una

etapa de mapeo donde se busca generar un conjunto de par llave,valor y finalmente una etapa de reduce encargada de operar los valores asociados a una misma llave. De forma gráfica podemos ver la figura 1 donde está la organización general de este método.

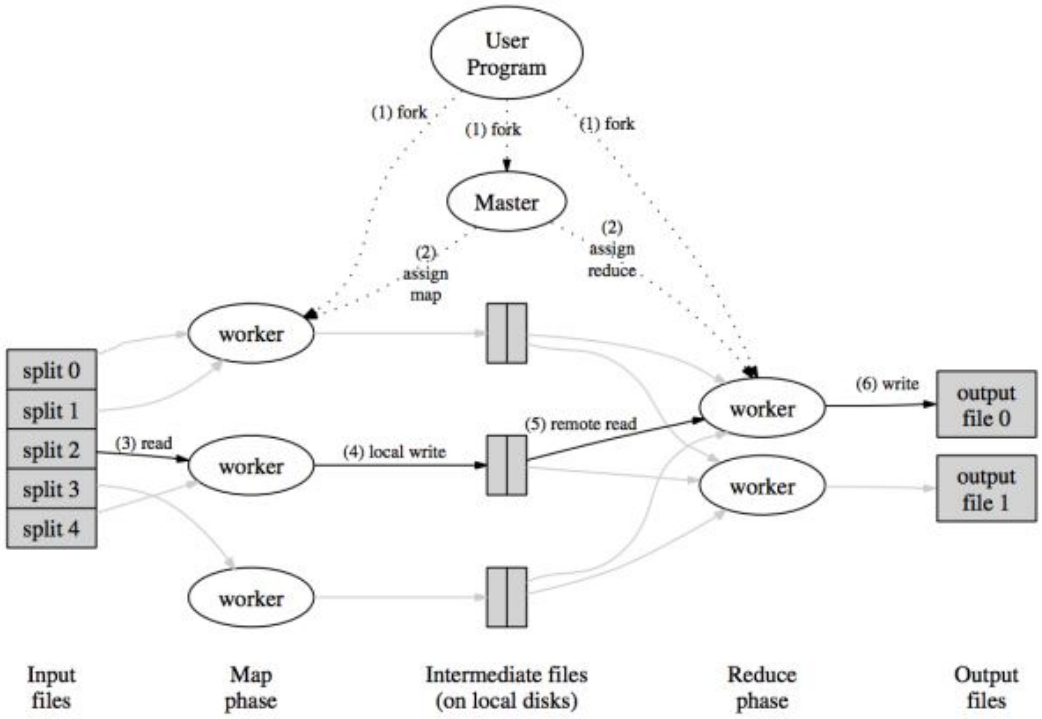

Figure 1. MapReduce

Para comprender mejor las etapas de mapeo y reduce se puede observar la figura 2. Donde básicamente se toma como llave la variable cust_id y por lo tanto la etapa de mapeo agrupará para cada llave todos los valores de amount asociados a la respectiva llave. La etapa de reduce recibe como entrada este par llave,valores y se encargará únicamente de sumar los valores asociados a cada llave.

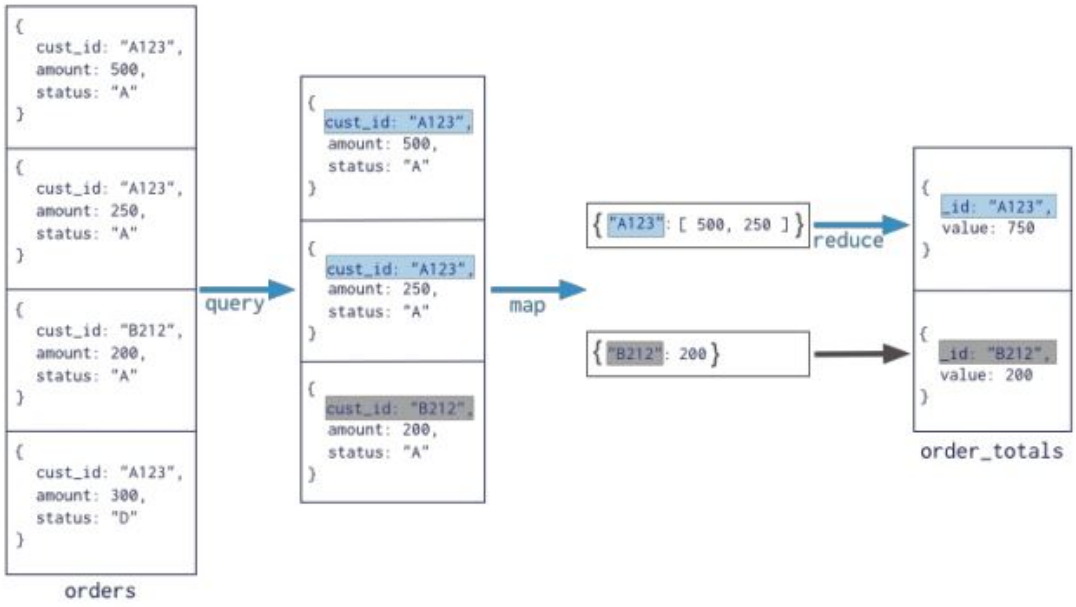

Figure 2. ejemplo práctico de MapReduce

IV. Enunciado

En este laboratorio, se pide que mejore el rendimiento de la solución propuesta en el laboratorio uno mediante el uso de hebras, entregando una solución capaz de leer un gran conjunto de datos, específicamente una base de datos chilena perteneciente a la municipalidad de Calbuco donde se almacena un registro de los permisos de circulación correspondientes al año 2022.

El objetivo es simular MapReduce creado por Google. Se debe utilizar hebras para representar a los worker de las etapas map() y reduce() tal como se puede ver en la figura 1, además de utilizar los archivos intermedios presentados en la figura 1.

IV.A. Cálculo de propiedades

Para este lab, las propiedades que se calcularán para cada columna son:

1. Indicar tasación total para cada grupo de vehículo
2. Indicar el total del valor pagado para cada grupo de vehículo.
3. Indicar para cada grupo de vehículo la cantidad de vehículos con dos y cuatro puertas.

IV.B. Lógica de solución

Se solicitan tres programas hechos en el lenguaje de programación c. Uno para coordinar y distribuir los datos de entrada a las diferente hebras de la etapa mapeo, además debe coordinar a las hebras de mapeo y reduce, ya que la idea es iniciar la etapa de reduce solo cuando haya finalizado por completo la etapa de mapeo, por otro lado, esta hebra coordinadora debe leer los archivos generados por las hebras de la etapa reduce para unir la información y entregar los resultados finales en un nuevo archivo de texto.

El segundo código, se encargará de representar la etapa de mapeo, la cual mediante hebras calculará las propiedades descritas previamente. La idea es que estas hebras reciban mediante la memoria compartida los datos de entrada y generen los archivos de salida correspondiente (ver salida de map en figura 2).

Finalmente el tercer código representa la tercera etapa, correspondiente a Reduce, por lo que cada hebra leerá los archivos de salida creados en la etapa map, para luego generar un archivo por propiedad (o bien un único archivo) indicando entonces los resultados parciales para cada llave (ver salida de reduce en figura 2).

IV.C. Coordinador

El primer programa debe crear n hebras para la etapa de mapeo y m hebras para la etapa de reduce, se espera que mediante el uso `pthread_create` se asigne a estas hebras el programa de mapeo o reduce a ejecutar.

Una vez creadas las hebras correspondientes, debe realizar la tarea de distribuir mediante el uso de estructuras, los datos a las diferentes hebras encargadas de mapear, es decir, leerá línea a línea el archivo de entrada para luego entregarle a las hebras de mapeo (mediante memoria compartida) la respectiva entrada. Otro camino es leer el archivo completo, guardarlo en un arreglo de estructuras (declarado como global) y proceder a recorrer este arreglo, escribiendo en la estructura dicha información. Una vez enviadas la cantidad de líneas totales a leer del archivo, enviará un mensaje de FIN a las hebras de mapeo.

Una vez distribuidos los datos, la hebra principal debe garantizar mediante el uso de `pthread_barrier_wait` que comience la etapa de reduce una vez finalicen las hebras de mapeo.

Por último, se encargará de leer los archivos de textos generados por las hebras de la etapa reduce, para luego escribir un archivo de texto final el cual tendrá los resultados finales. Es decir, le recibirá a la etapa reduce los resultados parciales de cada propiedad, por lo que la hebra coordinador operará esos resultados parciales para obtener el total.

La salida del archivo final debe tener el formato:

```
Total de tasaciones para Vehiculo Liviano: 23005623
Total de tasaciones para carga: 35889448
Total de tasaciones para Transporte Publico: 26654456
Total de valor pagado para Vehiculo Liviano: 10223665
Total de valor pagado para carga: 15632658
Total de valor pagado para Transporte Publico: 9654456
Total de vehiculos con 2 puertas para Vehiculos Livianos: 1000
Total de vehiculos con 4 puertas para Vehiculos Livianos: 3000
Total de vehiculos con 2 puertas para carga: 100
Total de vehiculos con 4 puertas para carga: 0
Total de vehiculos con 2 puertas para carga: 100
Total de vehiculos con 4 puertas para carga: 0
Total de vehiculos con 2 puertas para Transporte Publico: 0
Total de vehiculos con 4 puertas para Transporte Publico: 50
Total de vehiculos con 5 puertas para Transporte Publico: 10
```

IV.D. Programa Mapeo

El segundo programa es el encargado de realizar las operaciones de mapeo. En este caso tenemos tres objetivos por lo que el mapeo abordará estos tres casos a través de tres funciones (una para cada propiedad).

Para el caso de la primera propiedad, el mapeo se encargará de considerar tres llaves principales: Vehículo Liviano, carga y Transporte Publico. Para cada llave, almacenará un listado de las tasaciones que le corresponden a cada grupo, ejemplo:

- Vehículo liviano: 4550702, 2292467, 1322987, ...
- carga: 1, 2,3, ...
- Transporte Publico: 1,1,.....

Esto mismo se repetirá para obtener el valor pagado por grupo y la cantidad de puertas. La idea es que la salida sea un archivo de texto que contenga los valores asociadas a las llaves. Ejemplo:

```
Vehículo liviano: 4550702, 2292467, 1322987, ......
carga: 1, 2,3, .....
Transporte Publico: 1,1,.....
```

Para lo anterior, se espera que exista al menos una hebra por propiedad. En caso de que se ingrese una cantidad de hebras mayor a la cantidad de propiedades, se espera que distribuya de manera equitativa, por ejemplo: si son 3 propiedades y la cantidad de hebras para mapeo es 12, se puede tener 4 hebras por propiedad.

IV.E. Programa Reduce

Este código se encargará de tener las funciones de reducción, las cuáles en este caso son bastante simples.

1. Para el caso de las tasaciones y valores pagados, la función se encargará únicamente de sumar todos los valores asociados a la respectiva llave.
2. Para el caso de la cantidad de puertas, esta función debe contar la cantidad de repeticiones que hay para cada cantidad de puertas asociadas a un grupo de vehículos, tal como se indica en el ejemplo de archivo de salida mencionado previamente.

Por lo tanto, cada hebra reduce debe leer el archivo de texto que le corresponde y hacer el cálculo parcial, es decir, debe generar un archivo de texto por propiedad, indicando la suma parcial (). Ejemplo:

```
Total de tasaciones para Vehiculo Liviano: 23005623
Total de tasaciones para carga: 35889448
Total de tasaciones para Transporte Publico: 26654456
```

Una vez generados los archivos de salida, debe enviar un mensaje a la hebra coordinadora informando que ha finalizado.

Al igual que en mapeo, debe distribuir de manera equitativa las hebras y las propiedades a calcular.

El programa se ejecutará usando los siguientes argumentos (ejemplo):

```
./lab2 -i permiso-de-circulacion-2022.csv.csv -c 10000 -n 5 -m 4 -d
```

- **-i**: nombre de archivo de entrada
- **-c**: cantidad de líneas de archivo de entrada.
- **-n**: cantidad de hebras a crear para map.
- **-m**: cantidad de hebras a crear para reduce.
- **-d**: bandera o flag que permite indicar si se quiere ver por consola el resultado final de las métricas.

Ejemplo de compilación y ejecución:

```
>> make
>> ./lab2 -i permiso-de-circulacion-2022.csv.csv -c 10000 -n 5 -m 4 -d
```

Como requerimientos no funcionales, se exige lo siguiente:

- Debe funcionar en sistemas operativos con kernel Linux.
- Debe ser implementado en lenguaje de programación C.
- Se debe utilizar un archivo Makefile para su compilación.
- Realizar el programa utilizando buenas prácticas, dado que este laboratorio no contiene manual de usuario ni informe, es necesario que todo esté debidamente comentado.
- Que el programa principal esté desacoplado, es decir, que se desarrollen las funciones correspondientes en otro archivo .c para mayor entendimiento de la ejecución.

V. Entregables

El laboratorio es en parejas y se descontará 1 punto por día de atraso. Debe subir en un archivo comprimido a usachvirtual los siguientes entregables:

- `Makefile`: Archivo para make que compila el programa. De no incluirlo, el trabajo será evaluado con la calificación mínima.
- `coordinador.c`: archivo con el código del coordinador. Puede incluir otros archivos fuente. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.
- `mapeo.c`: archivo con el código para el cálculo de propiedades. Puede incluir otros archivos fuente. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.
- `reduce.c`: archivo con el código para el cálculo de propiedades. Puede incluír otros archivos fuente. Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje.
- Trabajos con códigos que hayan sido copiados de un trabajo de otro grupo serán calificados con la nota mínima.

El archivo comprimido debe llamarse: RUTESTUDIANTE1_RUTESTUDIANTE2.zip
Ejemplo: 19689333k_186593220.zip
NOTA: los laboratorios son en parejas, las cuales deben ser de la misma sección. De lo contrario no se revisarán laboratorios.

VI. Fecha de entrega

13 de Julio antes de las 23:59 horas.