Presentación

Objetivos

Esta práctica tiene como objetivo el diseño e implementación de una aplicación concurrente multi-hilo utilizando phtreads. Para su realización se pondrán en práctica muchos de los conceptos presentados en esta asignatura.

Presentación de la práctica

Hay que presentar los archivos con el código fuente realizado. Toda la codificación se hará exclusivamente en lenguaje C.

Enunciado de la práctica

El objetivo de la práctica es implementar una versión concurrente del paradigma map/reduce para el procesamiento masivo de datos (Big Data).

Al programa únicamente se le pasará el path de un directorio que contendrá todos los ficheros de entrada que hay que procesar, el path del directorio de salida en donde se escribirán los ficheros de salida y el número de reducers que hay que crear:

Sintaxis: ConcMapReduce <input dir> <ouput dir> [num_reducers]

Habrá que implementar dos versiones de la práctica: una en C utilizando pthreads y otra en java utilizando los threads de java. Implementar la aplicación para que soporte un número variable de ficheros de entrada y reducers. Por defecto, se utilizarán 2 tareas de reducción.

- **Paradigma MapReduce.**

MapReduce es un modelo de programación utilizado originalmente por Google para dar soporte a la computación paralela sobre grandes conjuntos de datos en clusters de ordenadores, utilizando hardware de propósito general.

El nombre del paradigma está inspirado en los nombres de dos importantes métodos de la programación funcional: Map y Reduce. El Map toma un conjunto de datos y los convierte en otro conjunto de datos, en el que los elementos se dividen en tuplas (pares clave/valor). En segundo lugar, la tarea reduce, toma un subconjunto de las claves generadas por todas las tareas de map como entrada y combina los datos tuplas en un conjunto más pequeño de tuplas (realiza una operación de reducción y acumula todos los resultados parciales de una clave en un único resultado). Como la secuencia de MapReduce el nombre implica, la reducción se realiza siempre después del map.

La principal ventaja de MapReduce es que es fácil de escalar procesamiento de datos en múltiples nodos. En el modelo MapReduce, el procesamiento de datos primitivos son llamados mapas y reductores. Descomposición de una aplicación de procesamiento de datos en mapas y reductores a veces es no trivial. Pero, una vez que se implementa una aplicación MapReduce, escalar la aplicación para que se ejecute en cientos, miles o incluso decenas de miles de máquinas en un clúster consiste en simplemente realizar un cambio de configuración, sin

necesidad de modificar el programa. Esta sencilla escalabilidad es lo que ha atraído a muchos programadores a usar el modelo MapReduce. MapReduce ha sido adoptado ampliamente gracias a que existe una implementación OpenSource denominada Hadoop.

En la Figura 1 podemos ver las diferentes etapas de ejecución de una aplicación MapReduce:

- **Etapa de División (Split)**: Esta etapa consiste en dividir los datos de entrada en diferentes particiones (splits), cada una de las cuales se asignará a un Map diferente para que sean procesados de forma paralela/concurrente. Para obtener las diferentes particiones se asume que cada fichero se procesará de forma independiente. Además, si el fichero es muy grande, este se puede dividir en diferentes particiones (en este caso el tamaño de partición suele ser de 64 MB).

- **Etapa de Mapeo (Map)**: Los Maps procesan todos los datos de entrada asociados con una partición. Los datos de entrada se pasan a la función Map línea a línea en forma de tuplas clave/valor. En este caso, la clave es el offset del fichero y el valor el contenido de la línea. Como resultado del procesamiento de cada tupla de entrada, la función de Map puede emitir cero o más resultados. Estos resultados, también son generados en forma de clave/valor y se pasan a la etapa de mezcla y ordenación.

- **Etapa de Mezcla y Ordenación (Suffle):** A cada salida de la función Map se le asigna una tarea de reducción mediante una función de partición. La función de partición recibe la clave y el número de reductores definidos por el usuario y retorna el índice del reductor asignado. La función de particionado por defecto se basa en obtener el hash de la clave y utilizar el hash módulo el número de reductores.

  Entre las etapas de mapeo y reducción los datos son mezclados (ordenados paralelamente / intercambiados entre nodos) con el objeto de mover los datos desde el Map/nodo donde fueron producidos hacia el nodo en el que serán reducidos. Al final de la etapa de mezcla y ordenación, cada uno de las tareas de reducción recibirá un subconjunto de las claves generadas y por cada clave todas las valores ordenados que se hayan emitido en cualquiera de los Maps para esa clave.

- **Etapa de Reducción (Reduce)**: La etapa de reducción procesa los resultados parciales generados por las tareas de Map. Se llama a la función de Reducción de la aplicación una vez para cada clave única. Los datos de entrada de esta etapa consisten en una tupla clave/lista de valores. En la lista, tenemos todos los *valores* emitidos con esa clave por la etapa de Mapeo. La Reducción puede iterar entre los valores que están asociados con esa llave y producir cero o más salidas. La salida de la etapa de reducción se escribe en un fichero y también tiene la forma de una tupla clave/valor. Cada tarea de reducción genera su propio fichero de salida.

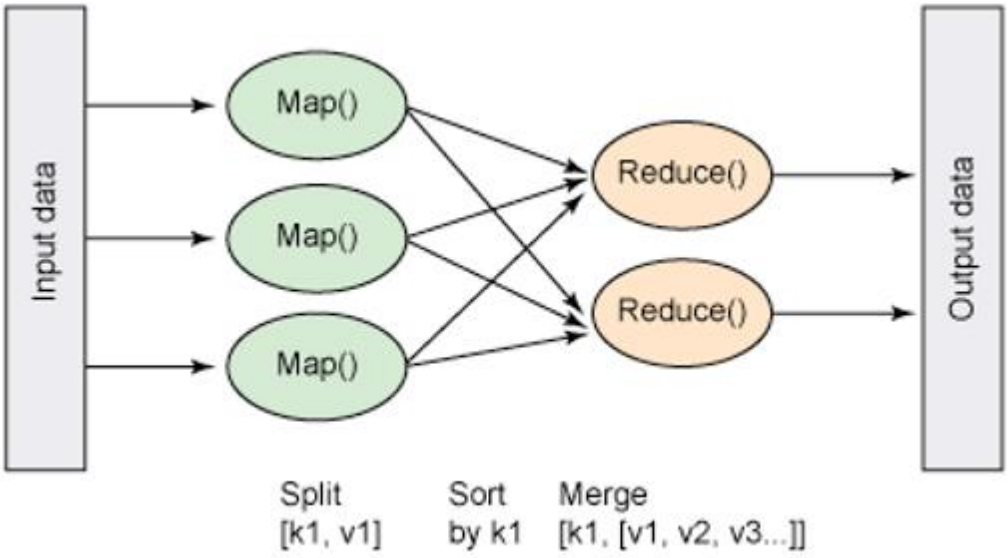

Figura 1. Etapas paradigma MapReduce

En la Figura 2 podemos ver un ejemplo de cómo se aplicarían las tareas de las diferentes etapas a una aplicación MapReduce *WordCount*, que cuenta el número de veces que aparece cada palabra en el texto de entrada. En esta aplicación la función de Map descompone la línea de texto (valor de la tupla) del fichero de entrada en palabras y emite para cada una de ellas una tupla con la palabra como clave y el número 1 como valor. La función de reducción recibe como entrada todos valores asociados a una clave generados por la etapa de mapeo, los suma (obteniendo la frecuencia de aparición de esa palabra en el fichero de entrada) y genera un resultado consistente en la palabra y el número de apariciones total.

En el ejemplo, la aplicación MapReduce procesará un fichero de entrada que contiene 3 líneas, las cuales se dividirán en 3 splits diferentes, que se asignarán a 3 tareas de Mapeo. A su vez, la aplicación consta de 4 tareas de reducción. A partir de estas condiciones iniciales, la aplicación de MapReduce se ejecutará de la siguiente forma:

1. La etapa de división particiona el texto de entrada en 3 splits, cada uno de los cuales es asignado a una tarea de Mapeo independiente.

2. Cada etapa de Mapeo procesa las líneas de su partición de entrada en forma de clave/valor. En el caso de la tarea de Mapeo 1, recibirá una única tupla de entrada {0, "Deer Bear River"}. La función de mapeo, descompondrá la línea de texto en palabras y para cada una de ellas emitirá un resultado intermedio en donde la clave será la palabra y el valor un 1. En el caso del Map1, emitirá los siguientes resultados: {Deer,1}, {Bear, 1} y {River, 1}.

3. En la etapa de mezcla y ordenación las claves asociadas con los resultados parciales emitidos por la etapa de mapeo ser reparten/particionan entre las tareas de Reducción. En el este ejemplo tenemos 4 tareas de Reducción. Por ejemplo, a la tarea de Reducción 1 se le asigna la clave Bear, por lo que en esta etapa recibe todos los resultados generados por las tareas de mapeo con esa clave ({Bear,1} por parte del Map1 y {Bear,1} por parte del Map3), los une y los ordena y genera como entrada para la tarea de reducción correspondiente una única entrada por clave con todos los valores generados para esa clave (en este caso: {Bear, {1, 1}}

4. Cada etapa de reducción recibe un subconjunto de las claves generadas por la fase de Mapeo y por cada clave una lista de todos los valores emitidos para esa clave. En el ejemplo, la tarea de reducción 1 recibirá como entrada la tupla {Bear, {1, 1}}, la cual

será procesa por la etapa de reducción sumando todas las ocurrencias de esa palabra (1's) y generando una tupla de resultado {Bear,2}, consistente en la palabra y el número total de apariciones de dicha palabra.

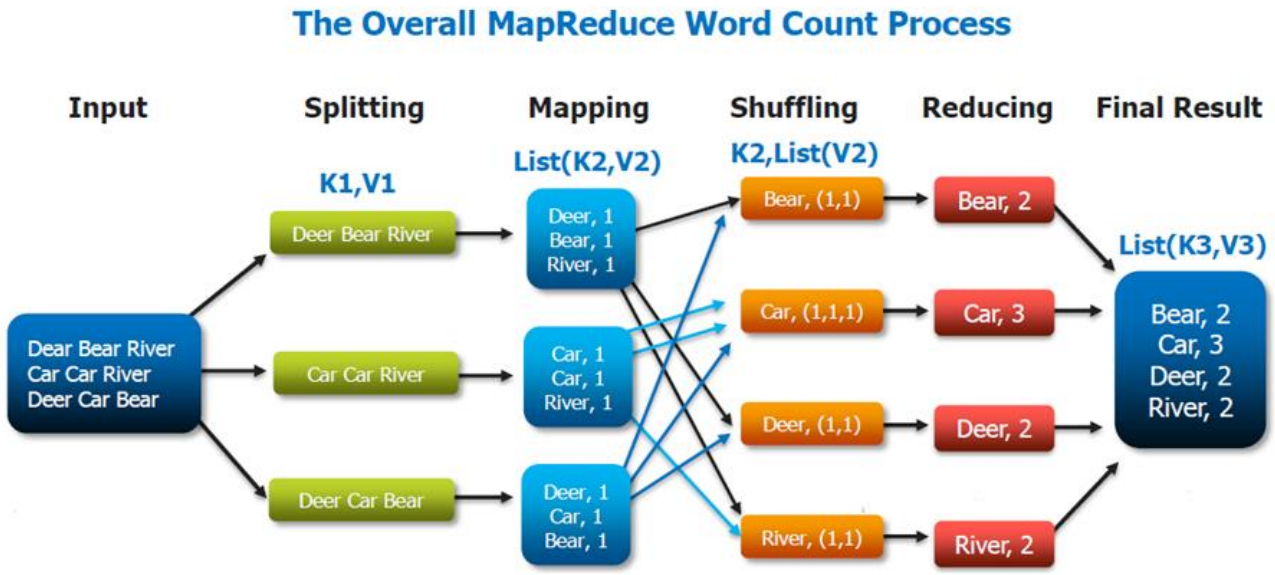

Figura 2. Ejemplo aplicación MapReduce para contar la frecuencia de aparición de las palabras en un texto.

- **MapReduce Concurrente.**

La implementación original del paradigma MapReduce, en Hadoop por ejemplo, está pensada para ejecutarse de forma paralela en un cluster de ordenadores y utilizando un sistema de archivos distribuidos como HDFS. Vuestra implementación de este paradigma, será concurrente y se circunscribirá a una sola máquina/proceso, de forma que los diferentes hilos que integran la aplicación se puedan comunicar mediante memoria compartida.

El objetivo de la práctica es que las diferentes etapas del paradigma MapReduce se ejecuten de forma concurrente utilizando múltiples hilos de ejecución. A continuación, os indicamos como se pueden paralelizar las diferentes etapas de la aplicación.

- *Etapa de División (Split):* En esta etapa paralelizaremos la lectura de los ficheros de entrada. Tendremos tantos hilos de ejecución como splits se puedan generar (al menos 1 split por cada fichero y para los ficheros muy grandes tendremos un split por cada 8MBs).

- *Etapa de Mapeo (Map):* En esta etapa crearemos un hilo de ejecución por cada tarea de Mapeo, ó lo que es lo mismo, un hilo por cada split. Cada hilo de Mapeo recibirá los datos (tuplas clave/valor) de su hilo de Split asociado mediante una lista en memoria compartida. Los resultados emitidos por el hilo de Mapeo se enviarán al hashmap asociado con la clave generada (aplicando la función de hash) y que será gestionado por el Hilo de Mezcla correspondiente.

- *Etapa de Mezcla y Ordenación:* En esta etapa, tendremos tantos hilos de ejecución como tareas de reducción. Cada Hilo de Mezcla recibirá en una lista con todos los resultados

parciales asociados con el rango de claves que tiene asignadas. Estas claves las juntará y ordenará, para posteriormente pasárselas a los hilos de Reducción.

- *Etapa de Reducción:* Tendremos tantos hilos de reducción como reducers haya especificado el usuario en el parámetro de entrada. La etapa de reducción no se puede completar hasta que las 3 etapas anteriores (Split, Map y Mezcla) se hayan completado en su totalidad. Cada hilo de Reducción recibirá un hashmap con las tuplas asociadas con las claves que tiene que procesar. Los resultados de la reducción se generarán directamente en fichero.

Los hilos de ejecución para las taras de Split, Map y Mezcla se puede ejecutar concurrentemente entre sí. Mientras que los hilos de ejecución asociados con las tareas de Reducción, para implementar la sincronización necesaria, no se crearán hasta que se haya completado las 3 etapas anteriores.

Funciones involucradas.

En la práctica podéis utilizar las siguientes funciones de c, entre otras:

- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);`
- `int pthread_join(pthread_t thread, void **retval);`
- `void pthread_exit(void *retval);`
- `int pthread_cancel(pthread_t thread);`

Análisis prestaciones

Calcular el tiempo de ejecución de la versión concurrente y compararlo con el de la versión secuencial. Discutir los resultados obtenidos. El objetivo de la práctica es que **la versión concurrente sea más rápida que la versión secuencial**.

Analizar también, cómo evoluciona el tiempo de ejecución en función del número de hilos utilizados en la aplicación concurrente, tanto para la versión C++, como para la versión java. Entregar un pequeño informe en donde se muestren (preferiblemente de forma gráfica) y expliquen los resultados obtenidos.

Indicar en el informe las características del hardware en donde habéis obtenido los resultados (especialmente el número de procesadores/núcleos).

Evaluación

La evaluación de la práctica se realizará en función del grado de concurrencia logrado. Podemos definir 5 grados de concurrencia:

- Lectura fichero concurrente. El objetivo es que todos los splits de entrada se lean de forma concurrente.
- Maps Concurrentes. Cada una de las tareas de Map de la aplicación se deben ejecutar de forma concurrente entre si. Tenemos tantas tareas de Map como splits.
- Reducers concurrentes. El usuario puede especificar el número de reducers que se tienen que crear/ejecutar en la aplicación. Estos reducers se tienen que ejecutar de forma concurrente entre si.
- Suffle concurrente: Si la estructura de control utilizada lo permite, la ordenación, unión y partición también se debe realizar de forma concurrente.
- Procesamiento datos en pipeline o flujo de datos: Siempre que las sincronizaciones no lo impidan se debería permitir tener diferentes datos procesándose en diferentes etapas. Es decir, no se tiene que ejecutar todas las etapas secuencialmente entre si (excepto la etapa de reducción). Por lo tanto, mientras la etapa de Split lee un trozo del fichero, el map puede estar procesado una línea anterior y el suffle estar ordenando los resultados parciales generados previamente.

En alguna etapa se puede necesitar sincronizaciones. En está práctica dichas sincronizaciones se implementará mediante los *join* y por lo tanto esperar a que un determinado hilo finalice.

Para que la aplicación se puede ejecutar concurrentemente, os debéis asegurar que las estructuras de datos utilizadas son thread-safe y que no generarán condiciones de carrera. En el caso de que no sea así, buscar una estructura alternativa que sea thread-safe.

Versiones Secuenciales

Junto con el enunciado se os proporciona la versión secuencial en C++ que implementa el paradigma MapReduce para la aplicación de contar el número de ocurrencias de cada palabra del fichero de entrada. Podéis utilizar dichas versiones para implementar las versiones concurrentes que se os pide en esta práctica.

- **Versión Quicksort en C.**

La versión secuencial en C se denomina WordCount.cpp y está implementada en C++. La versión adaptada para la práctica la tenéis disponible en el archivo comprimido Secuencial_WordCount.zip que os pasamos en la actividad.

Para poder utilizarla la versión C++, tenéis que descomprimir el archivo y compilarlo utilizando el *makefile* que viene con el fichero:

- make

Para ejecutar la versión secuencial, únicamente tenéis que especificar el nombre del fichero/directorio de entrada y la ruta del directorio en donde se guardarán los ficheros resultados.

- ./WordCount ./Test ./Output

También podéis obtener el tiempo requerido para la ejecución si utilizáis el comando time:

- time ./WordCount ./Test ./Output

Con estos ficheros también se os adjunta un conjunto de ficheros de entrada en el directorio Test, estos ficheros os pueden servir para probar la aplicación.

Formato de entrega

MUY IMPORTANTE: La entrega de código que no compile correctamente, implicará suspender TODA la práctica.

No se aceptarán prácticas entregadas fuera de plazo (salvo por razones muy justificadas).

La entrega presencial de esta práctica es obligatoria para todos los miembros del grupo.

Comenzar vuestros programas con los comentarios:

```
/* ------------------------------------------------------------------
Práctica 1.
Código fuente: WordCount.c
Grau Informàtica
NIF i Nombre completo autor1.
NIF i Nombre completo autor2.
------------------------------------------------------------------ */
```

Para presentar la práctica dirigiros al apartado de Actividades del Campus Virtual de la asignatura de Sistemas Concurrentes y Paralelos, ir a la actividad "Práctica 1" y seguid las instrucciones allí indicadas.

Se creará un fichero tar/zip con todos los ficheros fuente de la práctica, con el siguiente comando:

```
$ tar -zcvf prac1.tgz fichero1 fichero2 ...
```

se creará el fichero "prac1.tgz" en donde se habrán empaquetado y comprimido los ficheros "fichero1", fichero2, y ...

Para extraer la información de un fichero tar se puede utilizar el comando:

```
$ tar -zxvf tot.tgz
```

El nombre del fichero tar tendrá el siguiente formato: "Apellido1Apellido2PRA1.tgz". Los apellidos se escribirán sin acentos. Si hay dos autores, indicar los dos apellidos de cada uno de los autores separados por "_". Por ejemplo, el estudiante "Perico Pirulo Palotes" utilizará el nombre de fichero: `PiruloPalotesPRA1.tgz`

Fecha de entrega

Entrega a través de Sakai el 2 de noviembre, entrega presencial en la clase de grupo de laboratorio el dia 2 de noviembre.