Pontificia Universidad JAVERIANA Bogotá

Sistemas Operativos (SIU4085)

Proyecto 2- Marzo 2019

20%

Análisis Concurrente de un Log, usando una estrategia MapReduce

1. Objetivos del Proyecto

a. Resolver un problema concurrente utilizando procesos y los hilos de la librería POSIX.
b. Resolver el problema usando la estrategia MapReduce.
c. Utilizar mecanismos de comunicación o sincronización entre procesos o hilos para implementar el algoritmo.
d. Comparar el desempeño de todas las soluciones realizadas.
e. Familiarizarse con las llamadas al sistema para comunicación entre procesos vía pipes y sincronización de hilos usando los semáforos de la librería POSIX.
f. Familiarizarse con el uso de señales para comunicación asíncrona entre procesos e hilos.

2. Descripción General

Para la realización de este proyecto Ud. utilizará el proyecto anterior. Deben funcionar las dos versiones anteriores (hilos y procesos) siguiendo las especificaciones dadas (por ejemplo, deben tener un makefile, usar memoria dinámica, usar C estándar, debe funcionar en los laboratorios de la Universidad, etc.). Para esta última parte del proyecto Uds. van a comparar el desempeño de las dos primeras versiones, con la de una nueva versión producida por la extensión de los códigos anteriores en función de las especificaciones de este enunciado.

En la nueva versión las entradas, salidas y forma de hacer consultas sobre el archivo de log no cambian. Lo que cambiará es la implementación interna del algoritmo MapReduce. Una de las principales diferencias de esta implementación, con respecto a las anteriores, es que el número de mappers y reducers se creará una sola vez y, una vez creados, estos procesos o hilos se comunicarán a través de pipes. Si no se respeta esta especificación la solución estará incorrecta.

3. Procesos comunicándose vía Pipes

Las figuras 1 y 2 ilustran en el esquema de comunicación de procesos vía pipes.

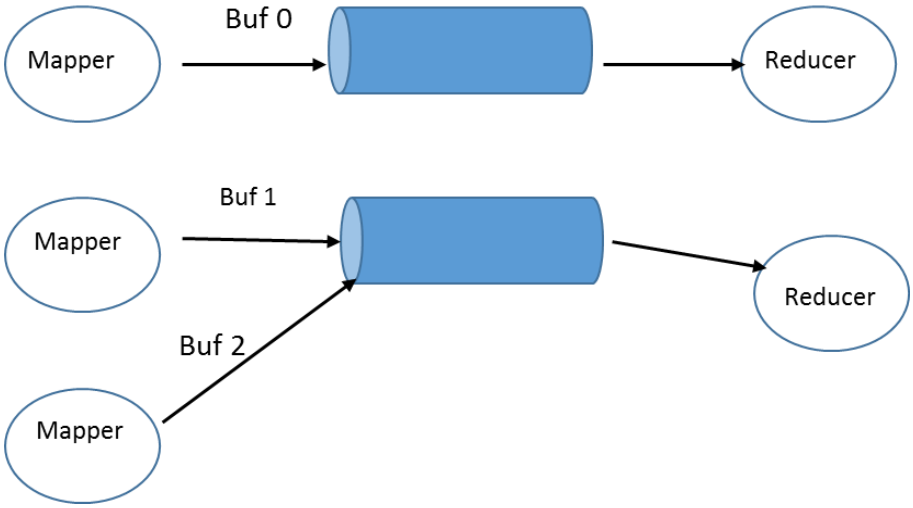

Figura 1: Comunicación entre Mappers y Reducers

Comunicación entre procesos: los procesos se comunicarán a través de pipes nominales o no nominales (es una decisión del grupo). Como pueden observar en la figura 2 hay, al menos un pipe entre el master y los mappers y entre los reducers y el master (son de color amarillo). Para la comunicación entre mappers y reducers también se deben crear pipes. En el dibujo se crean tantos pipes como reducers existen, pero Uds. pudieran tomar otra decisión, tal como: crear tantos pipes como mappers existan. Cualquiera que sea la decisión de diseño que el grupo tome, debe tener en cuenta que: a) Mappers y Reducers deben conocer por cual pipe van a escribir o de cual pipe van a leer; b) lo que se lee del pipe efectivamente se consume, desaparece del pipe; c) los reducers deben saber cuándo ha terminado la información que están escribiendo los mappers para una determinada consulta. Lo que se escribe por el pipe que conecta a los mappers con los reducers son los registros <clave, valor> que producen los mappers. Estos registros pueden escribirse de forma individual en el pipe, o en bloques, según son producidos por cada mapper. Por ejemplo, si escogen la opción de bloque el mapper que produce el Buf 0, debe escribirlos todos juntos en el pipe. El Master debe escribir la consulta a realizar en el Pipe 1 de la figura y debe leer los números que colocan los reducers en el Pipe 2 de la figura 2, que corresponden al número de registros que satisfacen la condición expresada en la consulta.

El Master: Como en el proyecto 1, el master va a leer el archivo de logs en una estructura de datos en la memoria, y dividirlo entre los procesos hijos. Antes de crear los hijos, el padre debe crear también todos los pipes de comunicación entre los procesos. Cuando cada mapper o reducer es creado, debe saber a qué pipe o pipes va a escribir o leer y cualquier otra información relevante, por ejemplo, los reducers deberían saber cuántos mappers escribirán a su pipe.

Una vez creados los procesos, el master seguirá el siguiente pseudo algoritmo:

1. *Muestra el menú al usuario*
2. *Lee la opción del usuario, si la opción es salir, ir al paso 7, sino voy a 3*

3. *Escribir la consulta en el pipe que comunica el master con los mappers, Pipe 1 en la figura, (recuerde que todo lo que se escribe en el pipe se consume, así que probablemente, si utiliza un solo pipe, deberá repetir la consulta para cada mapper)*
4. *Leer los resultados del pipe 2. Leer tantos registros como procesos reductores haya creado.*
5. *Mostrar los resultados al usuario*
6. *Volver a 2*
7. *Enviar una señal (signal) a los procesos mappers y reducers. Estos procesos culminarán al recibir la señal liberando las estructuras de datos utilizadas e imprimiendo el siguiente mensaje: Mapper con pid X termina.*
8. *El padre espera, usando la llamada al sistema wait, por todos los mappers y reducers creados.*

Mappers: la función del mapper no cambia con respecto a lo indicado en la primera parte del proyecto. Ellos buscan en el log y producen buffers con registros **<clave,valor>** como lo indica la figura 3 correspondiente a un ejemplo del enunciado anterior. Los mappers enviarán por los pipes, los pares <clave,valor>. A continuación, se muestra un pseudocódigo para un proceso mapper:

Mientras no reciba la señal

1.Leer la consulta del pipe 1.

2. Buscar en el trozo log que le corresponde, aquellos registros que cumplen con la consulta.

3. Enviar los registros encontrados (clave y valor) a través del pipe de comunicación con el reducer que le fue asignado.

Reducers: la función del reducer no cambia con respecto a lo indicado en la primera parte del proyecto. Ellos buscan los registros en el pipe que el master les asigna, leen los registros que cumplen la condición y los cuentan para producir un resultado al master. Es importante que los reducers sepan cuando terminan los registros que escriben los mappers. El resultado de cada reducer se escribe en el Pipe 2 de la figura 2. Este sería un pseudo algoritmo para un reducer que tiene el diseño de la figura 1. En este diseño cada mapper escribe todos sus registros en un solo bloque en el pipe y cada reducer sabe cuántos mappers van a escribir por el pipe en el que él es un lector:

Mientras no reciba la señal

1. *Desde 1 hasta el número de mappers que escriben por el pipe*
    a. *Leer buffer que escribe el mapper.*
2. *Contar los registros recibidos.*
3. *Enviar el número de registros que cumple la condición a través del Pipe 2*

Intermedios: si el valor de esta variable está en 1, los reducers imprimirán a un archivo todos los registros que van recibiendo por el pipe, indicando de alguna forma que reducer es el que los está escribiendo.

Fin de los Procesos: cuando el usuario indique el fin de los procesos, el master enviará una señal a los mappers y reducers. El master debe enviar la señal SIGUSR1 o SIGUSR2, y los mappers y reducer deben tener instalado el manejador correspondiente.

Figura 2: Comunicación entre el Master y el Resto de los Procesos

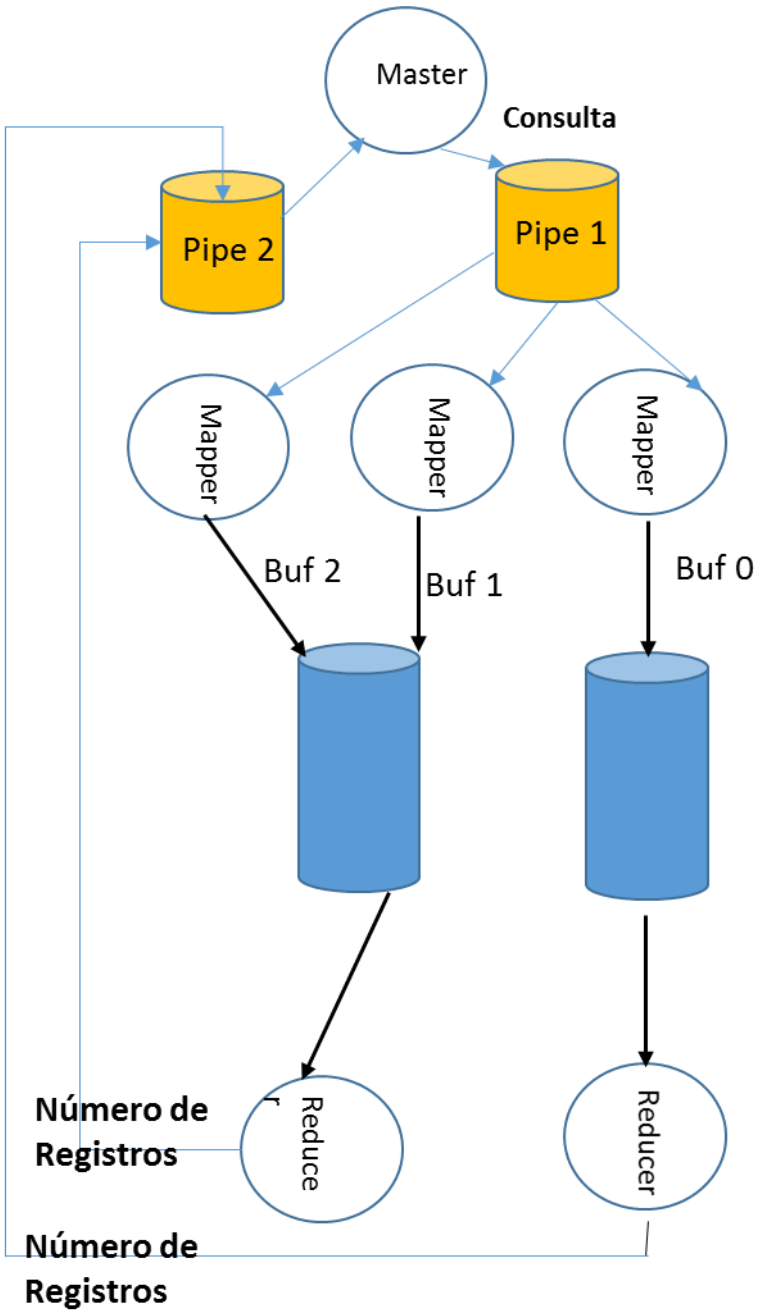

Figura 2: Comunicación entre todos los procesos

Buf 0

Clave	Valor
9	36
10	36
12	60

Buf 1

Clave	Valor
13	60
14	36
15	36
16	36
17	36
18	36
19	36

Buf 2

Clave	Valor
20	36
21	36
22	36

Figura 3: Clave y valor generado por los Mappers en los Buffers intermedios, de acuerdo a la consulta realizada (ejemplo del proyecto anterior)

El Informe

Todo el trabajo realizado es porque una empresa ha contratado sus servicios para implementar el algoritmo MapReduce por ello se desea que usted evalúe diferentes opciones de implementación con procesos o hilos. Para dar respuesta a la empresa Uds. deberán elaborar un informe donde se comparen las tres opciones implementadas, justificando con argumentos técnicos su implementación y presentando cuál es para usted la mejor opción.

El informe debe contener:

1. Cuáles son las 3 opciones a evaluar. Describa las principales características de cada opción.
2. Describa los elementos importantes de su diseño que se dejan libres en el enunciado de la comunicación por pipes: número de pipes, qué tipo de pipes se utilizaron, como se asignaron los pipes entre mappers y reducers, cómo se escriben los registros en el buffer o pipe que existe entre mappes y reducers, etc. Utilice dibujos que permitan entender mejor su diseño.
3. Compare **el desempeño de una consulta** en las tres opciones a medida que aumenta el tamaño del log. Los tamaños del log se encuentran en la siguiente tabla. Se usarán los mismos archivos del proyecto anterior.

La consulta que deberán hacer para todas las pruebas es: número de Jobs que usaron 2 o más procesadores en su ejecución: **5, >, 1**. Todas las pruebas se realizarán con 6 mappers y 2 reducers (2). Para las soluciones con procesos el valor de intermedios debe ser igual a 0.

	10000 registros	**20000 registros**	**30000 registros**	**40000 registros**
Procesos (los procesos se comunican a través de archivos, se crean y destruyen procesos en cada consulta)	Tiempo total de ejecución.	Tiempo total de ejecución.	Tiempo total de ejecución.	Tiempo total de ejecución.
Hilos (los mappers y reducers se crean solo una vez, se comunican a través de buffers compartidos)	Tiempo total de ejecución.	Tiempo total de ejecución.	Tiempo total de ejecución.	Tiempo total de ejecución.
Procesos (los mappers y reducers se crean solo una vez, se comunican a través de pipes)	Tiempo total de ejecución.	Tiempo total de ejecución.	Tiempo total de ejecución.	Tiempo total de ejecución.

Una vez que se llene la tabla con los valores requeridos para cada celda, se deben realizar gráficos donde se observe claramente el desempeño de cada solución para los diferentes tamaños de carga.

Compare el desempeño de las 3 soluciones. Justifique los resultados obtenidos, es decir analice las diferencias en el desempeño utilizando los conceptos vistos en la clase. Utilice argumentos técnicos para presentar la mejor solución para el cliente (su recomendación). Explique toda la terminología técnica usada.

Detalles de Implementación

1. La implementación se realizará en lenguaje C (C estándar). Para la creación de los hilos se utilizará la librería POSIX. Para estar seguro que el código es ansi c, debe utilizar el flag `-ansi` o el flag `-std=c89` con el compilador gcc.
2. Deben seguir en su código las recomendaciones de la Guía de Programación en C publicada en UVirtual (archivo estiloC.pdf, en la carpeta de información sobre el curso).
3. En su proyecto debe usar memoria dinámica para la creación de las estructuras de datos que manejarán los logs. Revise la llamada al sistema *malloc()*. Puede usar listas o arreglos creados en forma dinámica.
4. Debe validar llamadas al sistema, parámetros de entrada y los parámetros para realizar las consultas. Debe utilizar correctamente los recursos que solicita, es decir, no dejar procesos huérfanos o zombies, archivos abiertos, borrar archivos temporales después de que ya no sean necesarios, etc. En el caso de que en la ejecución de un programa se haya solicitado la permanencia de archivos intermedios, éstos deben borrarse al iniciar la siguiente consulta (es decir el profesor los debe poder verificar al final de cada consulta).
5. Deben hacer un makefile para crear los tres ejecutables.
6. Se les dará un log como ejemplo para que prueben su proyecto (ejemplo0). Su programa debe leer el archivo en el formato dado. El día de la sustentación la profesora les dará los archivos de prueba y éstos seguirán el mismo formato.

Observaciones Adicionales

- El proyecto lo deben realizar en grupos de 2 ó 3 estudiantes. No se permitirá la entrega de proyectos en forma individual, no serán calificados.
- Lo deben entregar el Jueves 16 de Mayo antes de las 1 pm por UVirtual. **Sin excepción**, los proyectos que no estén a esa hora en el UVirtual o en el buzón de correo del (a) profesor (a), no serán calificados.
- Para correr los casos para el informe no debe usar medidas tomadas en el enunciado del proyecto anterior y deben hacer aquellas modificaciones que el profesor y/o montor/a le sugirieron a su proyecto.
- La sustentación se realizará en horas cercanas a las horas de práctica. La entrega consiste de los códigos fuentes, el makefile y el informe con la evaluación de rendimiento. Todos los integrantes del equipo deben estar a la hora de la sustentación. Si alguno de los integrantes no se presenta a la hora de sustentación o llega después de concluido el periodo establecido para las sustentaciones, deberá pagar supletorio. Cualquier problema con el horario de la sustentación debe avisarse en los días previos a la misma.
- Los proyectos deben funcionar en las computadoras de los laboratorios de la Universidad. Si esto no ocurre y el equipo puede usar un computador personal para la sustentación, pero la nota final tendrá una penalización.
- Durante la corrección se podrá ejecutar el proyecto con archivo mayores a los especificados para el informe
- Cualquier duda sobre el enunciado del proyecto debe consultarla con los profesores en forma oportuna. La comprensión del problema y su correcta implementación, según lo indica el enunciado, es parte de lo que se está evaluando.
- Los grupos pueden discutir e intercambiar ideas de forma verbal, pero bajo ningún concepto pueden compartir código o resultados del informe. Si se detecta copia en los productos entregados, los integrantes de los grupos serán citados a la Dirección de Carrera y el caso será elevado al Decano de la Facultad.

Referencias

https://commons.wikimedia.org/wiki/File:Virginia_tech_xserve_cluster.jpg

Suerte

Profs. Mariela Curiel y Ricardo González.