



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: pthreads

Escape en Sistemas

13 de noviembre de 2014

Sistemas Operativos

Integrante	LU	Correo electrónico
Rodriguez, Pedro	197/12	pedro3110.jim@gmail.com
Benegas, Gonzalo Segundo	958/12	gsbenegas@gmail.com
Barrios, Leandro Ezequiel	404/11	ezequiel.barrios@gmail.com



Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

Contents

1	Implementación de Map-Reduce	1
2	Investigación	2
2.1	Explique cuál es el entorno y la situación donde se plantea el problema. (Motivación)	3
2.2	Identifique el problema	4
2.3	Identifique situaciones donde el problema se dispara y la consecuencias que esto genera.	5
2.4	Comente el background histórico del problema	6
2.5	Explique otros problemas asociados a la búsqueda de una mejor solución	7
2.6	Explique las soluciones propuestas	8
2.7	Evalúe soluciones propuestas	9
2.8	Discusión	10
2.9	Conclusiones	11

Abstract

Hoy en día, la globalización y creciente utilización mundial masiva de medios de información como la internet, ha impulsado la existencia de gigantes de la información. Ejemplos de estos son empresas como Facebook, Yahoo!, Google, Twitter y otros. La mayor parte del servicio provisto por estas empresas a sus usuarios, consiste y necesita de la utilización, procesamiento y análisis de grandes bases de datos. Por ello, una de las más frecuentes acciones que deben ser efectuadas es el filtrado de datos y su posterior procesamiento.

Un método comúnmente utilizado dentro del entorno de procesamiento de datos a través de clusters es el **MapReduce**, creado por la empresa Google. Este permite analizar un gran conjunto de datos, relacionándolos a través de un índice (*key*) a través de una operación **map**, para luego operar sobre los conjuntos de valores asociados a cada *key* a través de una operación **reduce**. Opcionalmente, es posible aplicar al conjunto de datos producido por **reduce**, una última operación **finalize**.

Considerando la inmensidad del volumen de información con el que se trabaja, y el gran costo económico que implican la fabricación y el funcionamiento de los servidores adecuados para manipularla, es importante lograr el máximo aprovechamiento de los recursos computacionales de los que estas empresas disponen. Por ello, se intenta trabajar con la máxima eficiencia algorítmica, temporal y de espacio posibles.

Parte I Implementación de Map-Reduce

Parte II Investigación

Para esta parte del TP se pide leer e interpretar el paper «**Job Scheduling for Multi- User MapReduce Clusters**»¹. Se sugiere además una serie de puntos para analizar el contenido del paper. Estos son los siguientes:

- 2.1 Explique cuál es el entorno y la situación donde se plantea el problema. (Motivación)
- 2.2 Identifique de problema
- 2.3 Identifique situaciones donde el problema se dispara y la consecuencias que esto genera.
- 2.4 Comente el background histórico del problema
- 2.5 Explique otros problemas asociados a la búsqueda de una mejor solución
- 2.6 Explique las soluciones propuestas
- 2.7 Evalúe soluciones propuestas
- 2.8 Discusión
- 2.9 Conclusiones

¹http://www.icsi.berkeley.edu/pubs/techreports/ICSI_jobschedulingfor09.pdf

2.1 Explique cuál es el entorno y la situación donde se plantea el problema. (Motivación)

La idea tras este paper surgió luego de que los autores se vieran frente a la tarea de diseñar un scheduler para MapReduce, para la empresa Facebook. Este sería el encargado de repartir el poder de cómputo en un *warehouse Hadoop*² de 600 nodos, en un entorno multiusuario, en donde se combinarían tareas de producción con tareas experimentales.

Este warehouse era utilizado para aproximadamente 3200 operaciones de MapReduce diarias, siendo que algunas de ellas eran tareas frecuentes de mantenimiento, análisis de datos, antispam y optimización, cuya ejecución se prolongaba a lo largo de horas, mientras que otras eran queries *AD-HOC* cuya ejecución demoraba unos pocos minutos. Por tal motivo, era de vital importancia poder lograr un mecanismo de **scheduling justo**.

²Hadoop es la versión *open-source* de MapReduce.

2.2 Identifique el problema

Se describen principalmente dos aspectos en donde el scheduling de un cluster MapReduce se diferencia de un mecanismo genérico de scheduling, y por los cuales el mecanismo de scheduling utilizado previamente presentaba una pérdida de rendimiento de entre 2 y 10 veces en comparación con el mecanismo presentado en el paper. Los dos principales aspectos problemáticos fueron la *localidad de datos*, y la *interdependencia entre las operaciones Map y Reduce*.

Localidad de Datos

Este problema comprende la necesidad inherente de la técnica MapReduce de tener acceso local a los datos con los cuales se están trabajando. Esta necesidad se debe principalmente a que a diferencia de los algoritmos “CPU- INTENSIVE”³, el MapReduce entra en la categoría de los denominados “DATA-INTENSIVE”, es decir, aquellos en donde el poder de cómputo necesario es despreciable frente a la gran cantidad de datos con los que se trabaja.

Desde esta perspectiva, contar con un scheduler eficiente en el acceso y la utilización de los datos, es mucho más importante que en otro tipo de situaciones. Todo esto está agravado por el costo que supone el almacenamiento, y la transmisión de datos entre distintos servidores. Este defecto se puede observar principalmente en **trabajos pequeños y/o concurrentes**.

Interdependencia

Se entiende por interdependencia a la necesidad de que todas las tareas de **map** se encuentren finalizadas al momento de iniciar la ejecución de **reduce**. Es fácil percibir la gran cantidad de inconvenientes que esto genera. Se mencionan particularmente los casos de **infrautilización de los recursos y starvation**, en donde se da que una única tarea adquiere cierta cantidad de slots, destinados a realizar la operación **reduce**, pero que se ven bloqueados/inutilizados hasta el momento en que esta tarea termine de realizar su operación de **map**, evitando de esta forma que otras tareas se ejecuten, y de **consumo excesivo de espacio de disco** destinado a los datos intermedios producidos por **map** (datos que no pueden ser liberados hasta que el trabajo termine).

³Los que necesitan un gran poder de cómputo.

2.3 Identifique situaciones donde el problema se dispara y la consecuencias que esto genera.

2.4 Comente el background histórico del problema

Hadoop está inspirado en el MapReduce de Google, por lo que gran parte de su implementación básica está fuertemente ligada a este proyecto. La primer solución al conflicto del scheduling, se encuentra provista por defecto dentro la propia implementación de Hadoop, y consta de una cola **FIFO** de 5 niveles de prioridad, de forma tal que cada vez que un slot se libera, el scheduler lo asigna a la más prioritaria de entre las tareas pendientes.

Sobre este enfoque, Hadoop aplica una **optimización de localidad**, al igual que lo hace MapReduce de Google: dado que una tarea generalmente consta de múltiples operaciones de **map**, luego de elegir una tarea el scheduler selecciona las operaciones map más adecuadas según un **criterio de localidad**. Serán elegidos entonces los maps que utilicen datos que se encuentren más cerca físicamente al worker que está corriendo la tarea. Es decir, serán elegidas primero las que se encuentren en ese mismo worker, luego las que se encuentren en el mismo rack, y finalmente las que se encuentren en racks remotos.

La gran desventaja del scheduler **FIFO** es el pésimo tiempo de respuesta para trabajos pequeños cuando se encuentran intercalados con trabajos grandes. Por ello, la primer solución que se implementó, es el mecanismo denominado **Hadoop On Demand (HOP)**. Este

2.5 Explique otros problemas asociados a la búsqueda de una mejor solución

2.6 Explique las soluciones propuestas

2.7 Evalúe soluciones propuestas

2.8 Discusión

2.9 Conclusiones