Sistema de Resolución de Matrices MTZ

Bourgeois Agustin, Coronel Renzo, Tolosa David

UTN Facultad Regional La Plata

Abstract. MTZ es un modelo distribuido para la resolución de operaciones de suma y resta de matrices con grandes dimensiones. En el presente trabajo realizaremos una descripción de la implementación de MTZ utilizando como base el lenguaje C y el protocolo TCP mediante la librería BSD Sockets.

Keywords: matriz, distribuido, suma, resta, TCP, C, BSD Socket.

1 Introducción

Procesamiento distribuido se define como, la forma en que es posible conectar distintas máquinas en cierto tipo de red de comunicaciones, generalmente una LAN o una red de área amplia o una red como Internet, logrando así, que una sola tarea de procesamiento de datos pueda ser procesada o ejecutada entre varias máquinas de la red, permitiendo una mejor utilización de equipos y mejora del balanceo de procesamiento dentro de una aplicación.

MTZ es un modelo formado por un servidor central que permite a los clientes que se conectan la realización de suma y resta de matrices con grandes tamaños de forma distribuida, repartiendo la carga de las operaciones de forma aleatoria entre los colaboradores o workers conectados al sistema.

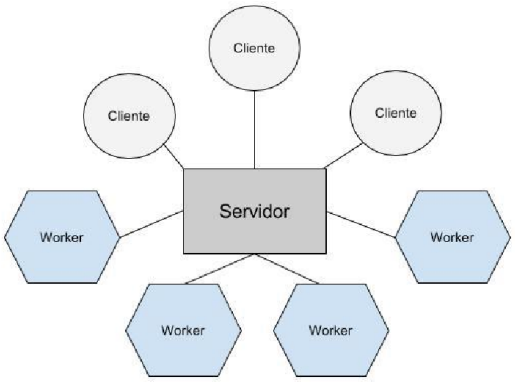

Fig. 1. Esquema general del sistema

2 Estado del arte

En la actualidad existe un modelo de programación utilizado por Google para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadoras conocido como MAPREDUCE. Este framework permite la resolución de aquellos problemas susceptibles de ser particionados y paralelizados, por este motivo las operaciones de álgebra lineal pueden ser resueltas de manera eficiente y eficaz con este método.

Las primeras aplicaciones de MapReduce fueron realizadas por Google para calcular el PageRank, donde multiplicaban matrices de gran tamaño para obtener la clasificación de páginas en una búsqueda.

Actualmente existen varias implementaciones de MAPREDUCE, algunas de ellas son Hadoop, CouchDB, MongoDB.

3 Motivación

La realización del presente trabajo nos permitió ser capaces de realizar una pequeña aplicación la cual resuelve un problema de forma cooperativa, siendo este un el puntapié inicial para entender los temas referidos a la distribución mediante una red y los conceptos de las interconexiones entre equipos.

Ya existiendo implementaciones de este tipo, fuimos capaces de realizar nuestra propia implementación aplicando gran parte de los conceptos aprendido a lo largo de la materia, los cuales fueron fundamentales para llevar a cabo el trabajo.

4 Propuesta

4.1 Objetivo

Realización de un sistema programado en C que permita el procesamiento distribuido de las operaciones suma y resta entre matrices.

4.2 Alcance

A efectos de centrarse en los aspectos importantes solicitados por la cátedra, hemos decidido realizar una implementación básica del sistema solamente con el lenguaje C y la Librería BSD Sockets abstrayéndonos de utilizar algunas otras herramientas de más alto nivel en la resolución del problema.

Para esto se implementa un protocolo de alto nivel basado en TCP el cual comunica las distintas partes del sistema. No se tendrá en cuenta la distribución de carga entre las partes, ni los fallos que se pudieran ocasionar por la falta de respuesta de una de las partes.

5 Implementación

5.1 Servidor

La aplicación servidor se implementó con Threads, se utilizó SQLite para crear un espacio compartido de memoria, está DB proporciona una manera liviana y simple de poder compartir información entre los distinto thread que la consultan sin tener que hacer uso de variables compartidas, evitando así, lidiar con todos los problemas de sincronización y álgebra de punteros que conlleva la comunicación entre los threads.

5.1.1 Lógica del servidor

El servidor escucha en un puerto a la espera de conexiones, luego de aceptar una conexión crea un Thread Cliente o Worker según el código de solicitud recibido y lo almacena en la BD como Cliente o Worker propiamente dicho.

Un Cliente, una vez conectado al servidor, puede solicitar la resolución de una operación de suma o resta. Cuando el servidor recibe una operación para resolver la descompone en suboperaciones, las cuales almacena en la base de datos para luego ser resuelta por los workers de la siguiente manera:

a11,...,a1j;b11,...,b1j;.

El cliente queda a la espera que los workers conectados al servidor resuelvan todas las suboperaciones.

Un Worker, luego de conectarse, solicita una suboperacion para resolver. El Thread del Worker conectado busca en la base de datos una suboperacion que no contenga resultado y no esté asignada a ningún worker y la envía al worker solicitante. Una vez calculada la operación el worker devuelve el resultado al servidor y este es almacenado.

Cuando el Thread del cliente comprueba que todas las suboperaciones de la operación solicitadas están resueltas, arma el resultado final y lo envía al cliente.

5.1.2 Base de datos

Como sistema de gestión de base de datos se eligió SQLite porque a diferencia de los sistema sde gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo.

El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones, esto reduce la latencia en el acceso a la base de datos debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son

guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

Fig. 2. Esquema DB

En la figura 2 se muestra el esquema de la base de datos. Las tablas cliente y worker contiene los id de los clientes y los workers conectados al servidor, la tabla suboperaciones contiene todas las suboperaciones de las operaciones solicitadas por los clientes. En esta tabla, el id_cliente es el identificador del cliente y se utiliza para saber a quién pertenece la suboperacion, cuando un cliente se desconecta del servidor se elimina su identificador de la tabla "cliente" y todas las suboperaciones relacionadas con dicho cliente. El id_worker se inicia como NULL cuando se carga una nueva operación y se actualiza su valor cuando un worker toma la operación para resolver.

5.2 Cliente

La aplicación cliente puede conectarse al servidor como un colaborador/worker para resolver suboperaciones o como un cliente para solicitar la realización de operaciones.

5.2.1 Lógica del Cliente

Si se ejecuta la aplicación como cliente, se muestra una pantalla de bienvenida al usuario con la ayuda necesaria para ejecutar las diferentes operaciones.

El Cliente realiza las siguientes actividades:

1. El usuario cliente almacena las matrices las cuales desea que sean operadas en los dos archivos <matrizA>.mtz y <matizB>.mtz.
2. Seguido a esto el usuario solicita mediante la consola de la aplicación la operación deseada <SUMA | RESTA> <file1> <file2>. Posteriormente estos archivos son parseados mediante el limitador "\n" y unidas ambas matrices por el delimitador ";". Siguiendo la siguiente estructura que es enviada al servidor:

a11, …, a1j\na21, …, a2j\n;b11, …, b1j\nb21, …, b2j\n

3. El Cliente se queda aguardando los resultados del servidor. Los cuales serán almacenados en un archivo <resultado.mtz> y visualizados en pantalla.

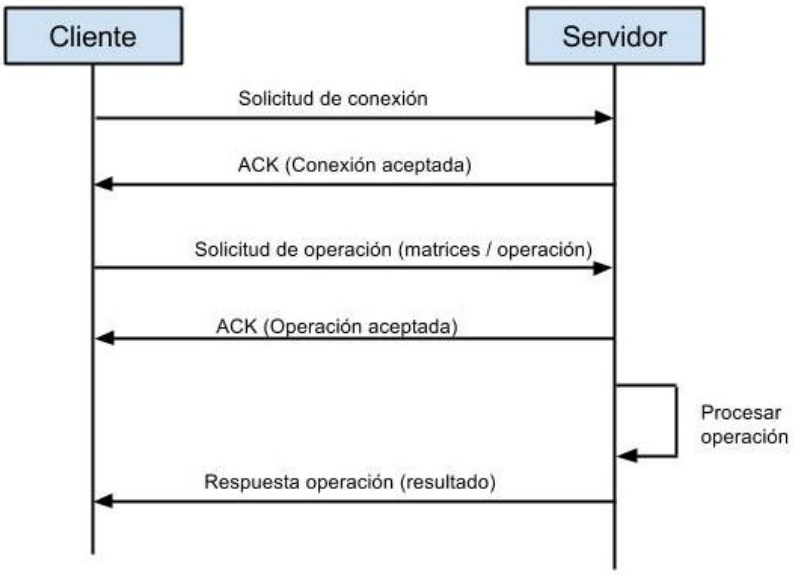

Fig. 3. Diagrama secuencia Server - Cliente

5.2.2 Lógica del Worker

El siguiente modo en que un cliente puede conectarse es como colaborador o worker, este solamente realiza una conexión con el servidor y queda a la espera de recibir trabajos a realizar. Los trabajos son recibidos con la siguiente estructura:

FilaA1;FilaB1 = a11,...,a1j;b11,...,b1j

Por simplicidad y a efectos de cumplir con el objetivo. Los worker solamente pueden resolver una operación por vez y hasta que el servidor no le envía el ACK de que recibió la resolución correcta, el worker no vuelve a solicitar un trabajo nuevo. Mientras que el servidor no tenga trabajos que realizar el worker se quedará en estado de espera y consultara con un intervalo de tiempo por la disponibilidad de nuevos trabajos para realizar.

De manera que un worker no recibía siempre todos los trabajos secuenciales. El servidor realiza la consulta a la bases de datos en intervalos de tiempo aleatorios. Además esta consulta ordena los registros del mismo modo, en forma aleatoria.

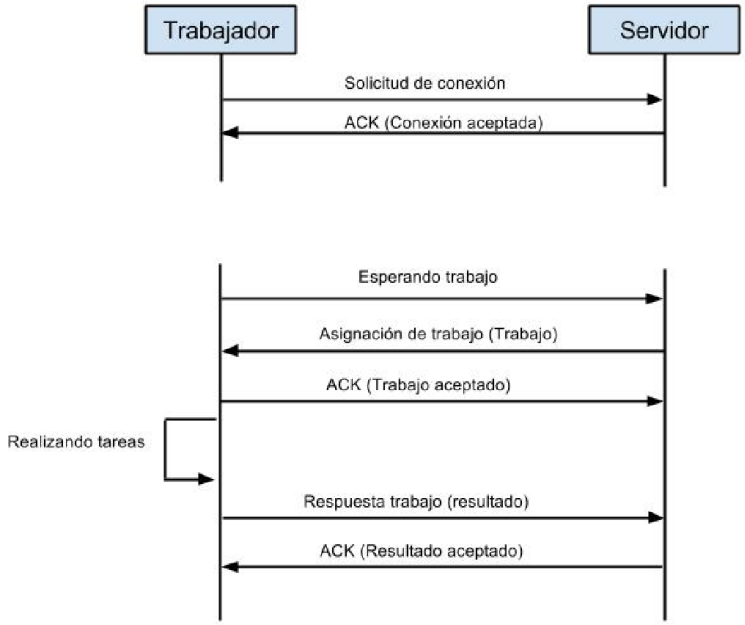

Fig. 4. Diagrama secuencia Server - Worker

6 Protocolo MTZ

Este protocolo está diseñado para poder realizar la comunicación entre las diferentes partes del sistema MTZ. Para este caso puntual se tomó como decisión utilizar en la capa de transporte TCP para así mantener las conexiones entre servidor y cliente. El protocolo MTZ está compuesto por la siguiente estructura que viaja dentro de la carga útil de TCP.

Código (8 bytes)	Length ()
Datos	

Descripción de los campos del mensaje :

- CODIGO : es un campo representado por 8bytes, que codifica los diferentes mensajes posibles a enviar.
- LENGHT : representado con un entero de 8bytes el cual indica la longitud total del mensaje (0...n byte) que es enviado.
- DATOS : representa la información a enviar, es de longitud variable y esta definida como una cadena de caracteres (char *) con delimitadores para facilitar su procesamiento.

Se implementaron funciones para leer los mensajes recibidos por las diferentes partes de la aplicación, estas funciones permiten obtener por un lado el header del mensaje que se recibió y por otro lado los datos contenidos dentro del mismo. También se implementó una función para armar el mensaje cuando algunas de las partes tenían información para enviar.

Dado que la longitud del mensaje es variable, determinado por el dominio del problema, fue necesario que estas funciones reservaran memoria de forma dinámica, crearán el mensaje y fueran enviados al destino. Del mismo modo, cuando es recibido un mensaje lo primero que se obtiene es el header, de longitud fija, que contiene el código y la longitud total (lenght). Esta longitud junto con el código me permite posteriormente recuperar los datos (lenght - header) donde viaja la información relevante.

7 Conclusión

Llevar a cabo el desarrollo de este trabajo, conllevo hacer uso de todos los conceptos aprendidos a lo largo de la materia. Fue necesario reforzar y a veces hasta aprender como manejar el lenguaje C. Muchas dificultades se nos presentaron dado la falta de conocimiento sobre el lenguaje mas que por una razon tecnica o conceptual.

Siendo MapReducr uno de los framework más conocidos y aplicados para este tipo de aplicaciones, es posible con algunos de sus conceptos, realizar un propio desarrollo simple que solucione un problema el cual se puede particionar en partes y posteriormente obtener un resultado.

Queda una puerta abierta para en un futuro poder agregar más operaciones a resolver, como multiplicaciones y resolución de sistemas de ecuaciones. Hasta quizas estudiar y aprender algún algoritmo que permita realizar una distribución de la carga de operaciones de forma equitativa entre los workers para evitar en lo posible que alguno de ellos se encuentre demasiado tiempo desocupado y minimizar así los tiempos de resolución.

Referencias

1. MapReduce – Wikipedia. URL: https://en.wikipedia.org/wiki/MapReduce
2. Socket programming and the C BSD API. URL: yolinux.com/TUTORIALS/Sockets.html
3. C/C++ and Dynamic memory allocation. URL: yolinux.com/TUTORIALS/Cpp-DynamicMemory.html
4. Biblioteca string ANSI C. URL: http://c.conclase.net/librerias/?ansilib=string
5. Función realloc ANSI C. URL: http://c.conclase.net/librerias/?ansifun=realloc