Reto 6

Procesamiento paralelo distribuido: Sistemas de recomendación basado en la Correlación de Pearson

07/11/2012

UNIVERSIDAD EAFIT

Luisa Fernanda Querubín Osorio

Jorman Andrés Bustos Gómez

# Definición del Problema

El problema es el cómputo de una matriz ce correlación que almacena datos de preferencias de los usuarios y los productos. Para entender mas de que se trata ver el Reto 5 Opción 2 que describe el problema de Correlación de Pearson.

## Requisitos Funcionales y no funcionales

### Funcionales

* El sistema debe ser ejecutado en un cluster de MPI.
* El algoritmo debe ser de ejecución paralela.
* El algoritmo debe retornar la matriz de correlación de la calificación de los usuarios a las películas.

### No funcionales

* El algoritmo debe retornar su tiempo de ejecución.
* El algoritmo puede ser ejecutado en clusters de MPI diferente tamaño.
* El algoritmo será ejecutado por consola.
* El algoritmo estará en lenguaje C y será ejecutado desde Linux para las pruebas.

# Identificación de entidades

Se identifican entidades iguales: El modelo a utilizar será con entidades que son iguales y se replican en el cluster de MPI el cual trabaja bajo el modelo de SIMD (Single Instruction Multiple Data)

# Modelo Matemático

La solución se presenta paso a paso en el Reto 6 para el algoritmo secuencial, la división de la matriz y el envío de los datos puede ser agregado para la operación de multiplicación de matrices que seria la tarea para paralelizar. Con Pearson

## Análisis Inicial – Pseudocódigo:

Entra: Mx, My, m;

GenMatrizUI(Mx,My); //valores aleatorios

CalcMatrizCorr(My,m);

CalcularSR(Mx, My);

* GenMatrizUI(fila,columna){

UI[fila][columna]

Para y=1 hasta My {

Para x = 1 hasta Mx{

UI[x][y] = random(0-5) //Random Float

}

}

}

* CalcMatrizCorr(My,m){

Corr[My][My]

Para a=1 hasta My {

Para u = 1 hasta My{

Corr[a][u] = Sumat1(a,u,m) / Raiz((Sumat2(a,m))\*Raiz(Sumat2(u,m)))

}

}

}

* CarcularSR(m, columna){

Para i=1 hasta My{

SR[i] = OrdenarColumna(i) //Ignorando el valor de [i] que es la diagonal

SeleccionarMayores(m)

}

}

* Sumat1(a, u, m)

Acum = 0

Para i=1 hasta m{

Temp = (UI[a][i] – Promedio(a)) \*(UI[u][i] – Promedio(u))

Acum = Acum + Temp;

}

* Sumat2(a, m)

Acum = 0

Para i=1 hasta m{

Temp = (UI[a][i] – Promedio(a))^2

Acum = Acum + Temp;

}

* Promedio (a)

Acum = 0

Para i=1 hasta m{

Temp = UI[a][i]

Acum = Acum + Temp;

}

Acum / m

* Es importante una elección de un buen algoritmo de ordenamiento, existe uno que busca todo el arreglo por los mayores, que seria útil dado que solo buscamos m mayores y dado que m << que My.
* Es posible ir utilizando los valores de Corr a medida que se obtienen? O es necesario esperar por toda la columna para realizar la búsqueda?
* Se paraleliza por datos, modificando los algoritmos para que cada maquina trabaje solo sobre una porción de la tabla.

# PCAM

Particionado: Se hará particionado de datos, la matriz será descompuesta en cantidades iguales según el número de computadores en el Cluster de MPI. Todos las maquinas simuladas incluida la “padre” (encargada de dividir los datos) procesaran una parte de la matriz.

Comunicaciones: Se hará con mensajes send/recieve de MPI para simular el cluster de multicomputador. La idea es que sean mensajes de uno a todos, bloqueantes hasta el recieve.

Aglomeración: El computador principal recibe los datos con un mensaje recieve que los almacena de forma temporal hasta que reciba todas las respuestas, para luego organizarlas

Mapeo: Los datos se dividen equitativamente según el tamaño de la matriz y el numero de maquinas. Tratando que se hagan en porciones pequeñas que no tarden mucho en enviarse pero que no sean muy pequeñas para evitar muchísimos envíos.

# Paso de Mensajes

Idle: La idea es que las maquinas no estén mucho tiempo en estado de espera para esto se trata que la porción de datos enviada no sea muy grande.

Block: las conexiones se bloquean mientras se envían los datos y se reciben las respuestas. Sin embargo, el “padre” debe tener un hilo alterno que escucha por las respuestas en caso que algún “hijo” acabe mientras esta enviando los datos a los demás.

Buffer: El buffer se utiliza para la recepción de los datos en cada maquina, es necesario pues se necesitan los datos completos para el inicio del procesamiento. (De pronto un buffer de algún límite, pues se da el escenario de recibir varias filas o columnas el nodo puede iniciar a procesar mientras recibe la segunda parte de la información)

El mensaje: Sera de tipos básicos de “arreglos de números” float.

Deadlock: se trata de evitar deadlocks mediante la distribución de las tareas con instancias de un mismo programa, de tal forma que una vez distribuidos los datos no se presente la necesidad de un mismo recurso para trabajar.

## Granularidad

De los datos debe tener un punto medio. Si es una matriz de 10.000x100.000. Esta se debe dividir por cantidades iguales según los equipos simulados por el cluster.

## Concurrencia

No existen problemas de concurrencia ya que las operaciones sobre la matriz se realizan de forma independiente y sin modificar los datos de la misma. Los resultados se ordenan una vez llegan al nodo padre luego de ser procesados

# Middleware

El middleware por el cual daremos solución serán las librerías de MPI.