Escuela de Informática.
Departamento de Informática y Computación.

Anexos adicionales.

Modificaciones realizadas al proyecto.

Asignatura: Computación Paralela.
Profesor: Oscar Magna Veloso.
Alumnos:
Sergio Abarca Flores.
Rosa González Soto.
Diego Hernández García.
Roberto Oñate Piedras.

Martes, 5 de Mayo de 2015.

Introducción

En los últimos anexos entregados el día martes 28 de abril del presente año, mostraron las mejoras que presentaba la solución que no estuvo implementada en el informe final. Hasta ese momento, la última solución era la mejor, pero en la entrega surgió una nueva idea.

Esta idea surge al combinar los métodos utilizados en las dos soluciones anteriormente entregadas, en decir, dividiendo de manera física el archivo CSV que contiene toda la información (primera solución) en los rangos dinámicos que se generan al ingresar la cantidad de procesos a utilizar para ejecutar el programa (segunda solución).

Con esta nueva solución, fue necesaria su creación e implementación en la plataforma clúster montada en el laboratorio número 5, para así realizar las mediciones y obtener sus métricas que nos permitirán concluir cuál de las tres soluciones generada es la óptima para el problema.

Cabe mencionar que para nombrar los anexos siguientes, se mantendrá la numeración utilizada en los últimos anexos entregados. El último anexo escrito en aquel informe fue

"Anexo I: Análisis y conclusiones."

Por lo tanto, el primer anexo comenzara con la letra J.

Tabla de contenidos

Anexo J: Código tercera solución.

```
from mpi4py import MPI
importcsv
import random
import sys
importos
from time import time

comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.size
name = MPI.Get_processor_name()


# Funcion que genera un voto en base a la probabilidad que tiene la comuna
# es por esto que se entregan como rango1 el porcentaje que tiene el
# partido de izquierda y rango2 como las suma de la probabilidad de
# partido de izquierda junto con la probabilidad del partido de derecha.

def Voto(rango1, rango2):
probabilidad = random.randint(1, 100)
voto = 0
    # rango perteneciente a partido de izquierda.
if probabilidad <= rango1:
voto = 1
    # rango perteneciente a partido de derecha.
elif probabilidad > rango1 and probabilidad <= rango2:
voto = 2
    # rango perteneciente a partido independiente.
else:
voto = 3
return voto

# Funcion que lee el archivo csv que contiene las comunas y
# sus probabilidades de cada partido politico. Estos datos se
# trasladan a una lista para reducir el tiempo de lectura de
# los datos al generar los datos con la funcion Voto


defLeerComunas():
    # Apertura de archivo.
reader = csv.reader(open('Data/comunas.csv', 'rb'))
lista = []
fori, row in enumerate(reader):
# Particionar la informacion con el delimitador ";"
particion = str(row[0]).split(";")
comuna = particion[0]
region = particion[1]
derecha = particion[2]
izquierda = particion[3]
independiente = particion[4]
dato = [comuna, region, derecha,
izquierda, independiente]
lista.append(dato)
return lista

# Funcion que lee la informacion entregada por SERVEL
```

```
# comparando una comuna con sus probabilidades y utilizando
# lafuncion Voto. Teniendo el voto generado, se asigna al
# partido politico.


defcontarVoto(comunas, archivo, rank, name):
wt = MPI.Wtime()
suma = 0
sumb = 0
sumc = 0
reader = csv.reader(open('TMP/' + str(archivo) + '.csv', 'rb'))
fori, row in enumerate(reader):
particion = str(row[0]).split(";")
comuna = particion[1].rstrip()
for j in range(len(comunas)):
# Se busca que la comuna leida en el archivo "data_server_'xx'"
            # coincida con la lista de comunas, para asi, obtener sus
            # probabilidades.
if comuna == comunas[j][0]:
                # Generar rango partido Izquierda
                rango1 = int(comunas[j][2])
                # Generar rango partido Derecha
                rango2 = (int(comunas[j][2]) + int(comunas[j][3]))
                # Guardar el voto generado con la funcion Voto
voto = Voto(rango1, rango2)
                # Acumular los votos de cada partido politico donde:
                # suma guarda los votos de izquierda
                # sumb guarda los votos de derecha
                # sumc guarda los votos independiente.
if voto == 1:
suma = suma + 1
elif voto == 2:
sumb = sumb + 1
else:
sumc = sumc + 1
sumtotal = suma + sumb + sumc
wt = MPI.Wtime() - wt
lista = dict(
izquerda=suma, derecha=sumb, independiente=sumc, total=sumtotal)
comm.send(lista, dest=0)


# Funcion que genera los archivos CSV para entregarlos a
# cada procesador.


defHacerCsv(nombre, data):
writer = csv.writer(open("TMP/" + str(nombre) + ".csv", "wb"))
for l in data:
particion = str(l).split(";")
particion[1] = particion[1].rstrip()
particion[0] = particion[0].rstrip()
writer.writerow([particion[0] + ";" + particion[1]])


def main():
if rank == 0:
print "*****Paralelo Granularidad Fina de CodigoVersion 2*****"
tiempo_inicial = time()
        # Catidad total de datos
```

```
total = 12905887
        # Definicion de los rangos para la creación de archivos CSV
rango = int(total / (size - 1))
        # Datos que seran agregado al ultimo nodo
resto = total - (rango * (size - 1))
process = 0
        f = open('Data/data_total.csv', 'r')
        p = f.readlines()
fori in range(size - 2):
inicio = rango * i
fin = rango * (i + 1)
process = process + 1
primera = p[inicio:fin]
HacerCsv(process, primera)
inicio = fin
fin = inicio + rango + resto
process = process + 1
primera = p[inicio:fin]
HacerCsv(process, primera)
fori in range(1, size):
comm.send(i, dest=i)
if rank != 0:
archivo = comm.recv(source=0)
comunas = LeerComunas()
contarVoto(comunas, archivo, rank, name)
ifrank == 0:
izquerda = 0
derecha = 0
independiente = 0
totales = 0
fori in range(1, size):
voto = comm.recv(source=i)
izquerda = izquerda + voto["izquerda"]
derecha = derecha + voto["derecha"]
independiente = independiente + voto["independiente"]
totales = totales + voto["total"]
tiempo_final = time() - tiempo_inicial
print " Izquierda: " + str(izquerda)
print " Derecha: " + str(derecha)
print " Independiente: " + str(independiente)
print " Votos totales: " + str(totales)
print "Tiempo total de ejecucion: " + str(tiempo_final)
        # Eliminacion de archivosgenerados
fori in range(1, size):
archivo = 'TMP/' + str(i) + '.csv'
os.remove(archivo)
return 0
main()
```

1. Explicación código

El código mostrado en las páginas anteriores, elimina laslimitacionesde las dos soluciones presentadas en informes anteriores (informe final y anexos).

Cuando se ejecuta la instrucción:

```
$mpirun -np (cantidad procesador) –hostfile (ruta del archivo hostfile) pythoncod_paralelo_total_v2.py
```

Posiblemente se disponga de una menor cantidad de procesadores, es por esto que lasolución se encarga de dividir la información de manera física, pero primero para realizar esto, es necesario que todos los datos a procesar estén en un solo archivo CSV.

El programa lee la cantidad de procesos y divide en n-1 (n: cantidad de procesadores) generando la cantidad de datos que tendrá cada archivo CSV nuevo. En caso de que la división diera decimales, solo se toma la parte entera para los n-2 archivos, el último archivo tendrá los datos que le corresponden por rango además de los datos faltantes.Se divide en n-1 porque el primer procesador se encarga de distribuir y recopilar la información entregada por el resto de los procesadores, por ende, no realiza cálculos sobre el archivo CSV.

Al igual que en la última solución presentada, una de sus grandes ventajas es la posibilidad de medir cual es la cantidad óptima de nodos que se deben utilizar en base a los tiempos de mejora que se presentan.

Las funciones utilizadas para generar los votos, obtener las probabilidades y leer las comunas, se mantienen en su forma. En esta nueva solución se agrega la función de crear archivos CSV y además dentro del main se encuentran las líneas de código que eliminan los archivos creados.

Anexo K: Tiempos.

1. Promedios.

Numero de procesos	Segundos
5	115,025
10	95,562
15	77,186
20	68,661
25	66,524
30	66,739
35	65,876
40	63,021
45	77,088
50	95,511
55	98,184
60	96,921
65	99,836
70	97,503
75	97,686
80	99,536

Tabla 1: Primera medición de tiempos, Elaboración Grupal.

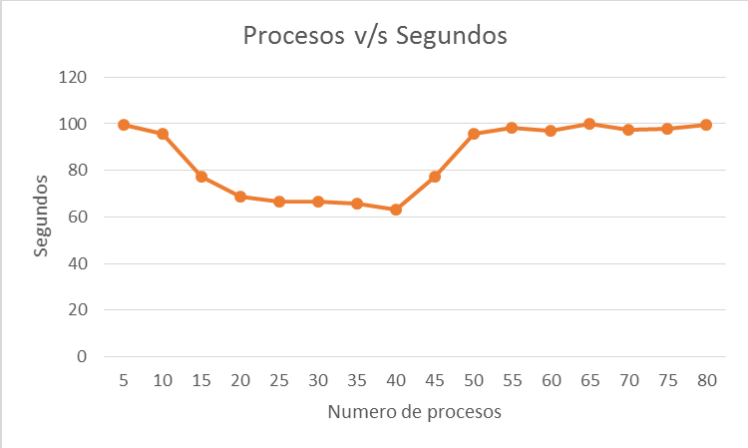

Grafico 1: Primera medición de tiempos, Elaboración Grupal

2. Observaciones.

En esta oportunidad, se tomaron 3 muestras de tiempo, con eso se calculó un promedio y ese es el dato que muestra el grafico 1. En este caso el ruido generado por el switch fue mínimo, porque las medidas se tomaron teniendo encendido solo los 5 computadores que forman el clúster y además que ningún computador estaba en funcionamiento, ya que las mediciones se hicieron de manera remota en la noche.

Anexo L: Métricas.

Antes de realizar las mediciones correspondientes, es necesario rescatar la siguiente información que deriva del anexo H: Métricas (Página 11)

- Tiempo secuencial: 250,994252 (Seg.)
- Mejor tiempo paralelo solución anterior: 14,5342152 (Seg.)

Los nuevos tiempos entregados por la granularidad gruesa son:

- Mejor tiempo paralelo nueva solución: 63,021 (Seg) utilizando 40 procesos.
- Peor tiempo paralelo nueva solución: 115,025 (Seg) utilizando 5 procesos.

1. Speed Up

El primer speed Up queda representado de la siguiente manera:

$$SpeedUp_{N.solucion} = \frac{Tiempo\ secuencial}{Mejor\ tiempo\ paralelo\ nueva\ solucion} = \frac{250{,}994252}{63{,}021} = 3{,}982708$$

Esto representa una mejora porcentualmente de 398,27% equivalente a decir que la mejora es casi de 3 veces más, comparado con el mejor tiempo encontrado en programa secuencial.

El segundo Speed Up, se compara el la mejor solución que se tenía del algoritmo paralelo granularidad fina.

$$SpeedUp_{N.solucion} = \frac{Mejor\ tiempo\ paralelo\ solucion\ anterior}{Mejor\ tiempo\ paralelo\ nueva\ solucion} = \frac{14{,}5342}{63{,}021} = 0.2306$$

Comparando la mejor solución que se tenía, esta nueva solución no representa una mejora, por el contrario, la solución anterior es casi 4 veces mejor que esta nueva solución

2. Eficiencia.

$$Eficiencia = \frac{Speed\ Up\ nueva\ solucion}{cantidad\ de\ procesos} = \frac{3{,}9827}{40} = 0{,}0996$$

Al comparar los resultados con los mostrados en el anexo anterior (Pagina 12) podemos comprobar que la eficiencia mostrada por esta solución no es la óptima, debido a la gran cantidad de ejecuciones que se realizan al generar un archivo CSV.

Anexo M: Análisis y conclusiones.

Luego de analizar los resultados de esta última solución paralela, se pudo observar que durante los primeros 40 procesos, los tiempos bajan, partiendo de los 115,025 segundos con 5 procesos hasta llegar a los 63,021 segundos con 40 procesos. Pero de ahí en adelante, comienza un aumento de los tiempos tendiendo a mantenerse cerca de los 95 segundos. Por lo tanto, este algoritmo nos muestra que en 40 procesos está el tiempo óptimo y esta idea que se ha mostrado en las otras dos soluciones.

Al comparar los datos obtenidos con la mejor solución que se tenía, la nueva solución no presenta mejoras, al contrario, es casi 4 veces más deficiente que la solución anterior. Este aumento en los tiempos de procesamiento se debe a la creación de archivos y también la eliminación de estos. Además el tiempo de comunicación desde que el nodo maestro envía la información a los nodos esclavos.Por lo mismo se demuestra que la segunda solución es la más efectiva y óptima para este problema.

La proyección de estos datos nos muestra un comportamiento lineal frenado, es decir, no aumenta ni bajan los tiempos de ejecución de manera significativa.