Escuela de Informática.
Departamento de Informática y Computación.

Informe de creación de plataforma paralela

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

Jueves, 23 de abril de 2015.

Resumen

En el siguiente documento se tratará el tema de creación de una plataforma clúster para realizar procesos paralelos del tipo SIMD, Single Intruction Multiple Data, utilizando datos reales obtenidos desde SERVEL a través de su página web. En este informe se mostrará la evolución del proyecto de análisis de tendencia política para las elecciones 2017, que comienza en 10 horas de procesamiento de datos de manera secuencial, hasta las 4 segundos que toma el procesamiento en paralelo para 200.000 datos de los 12.905.887, junto con sus estadísticas, Carta Gantt, y métricas correspondientes.

Abstract

The following document will be about the creation of a cluster plataform for SIMD, Single Instruction Multiple Data, for parallel processing, the information was obtained from the web page SERVEL. This report will be showing the evolution of the project analyzing the political trend for the next election 2017, this process will initially take 10 hours in secuential mode and it will evolve to a 4 seconds parallel processing for a data of 200.000 of 12.905.887, it will also include the statistics, Gantt chart and its metrics.

Contenido

1. Introducción

Con los grandes volúmenes de datos que manejan distintos sistemas informáticos actualmente, se hace indispensable contar con una máquina de alto rendimiento que sea capaz de procesar grandes cantidades de información y suplir las necesidades para las cuales está destinado. Para este cometido, la computación paralela entrega una solución efectiva y rápida que resuelve la problemática de procesamiento de enormes cantidades de información. Cabe mencionar, además, que el costo asociado a este tipo de implementación es muy bajo debido a su flexibilidad, ya que pueden tener todos los ordenadores la misma configuración de hardware y sistema operativo, diferente rendimiento pero con arquitecturas y sistemas operativos similares, o tener diferente hardware y sistema operativo, lo que hace más fácil y económica su construcción.

Para el caso particular del presente informe se desea realizar un estudio estadístico para indicar la tendencia política de las siguientes elecciones 2017. A partir de lo anterior, con el supuesto del 100% de los votantes, con los datos del Servicio Electoral de Chile (SERVEL), se definirá cuál es la tendencia política para las próximas elecciones, utilizando información correspondiente a las tendencias políticas de cada comuna a nivel nacional del año 2013. Cabe destacar, que las personas que cumplen 18 años entre los años 2014-2017 no están contempladas en este universo de simulación.

2. Planteamiento del problema

Para poder determinar la tendencia política de próxima elección presidencial 2017 se requiere de la información de qué tendencia política tuvo cada comuna en cada región de Chile para poder hacer una determinación estadística. Esta información se obtiene a través la web del SERVEL de Chile, que es un organismo autónomo encargado de organizar, fiscalizar y supervigilar todos los actos electorales prescritos por la Constitución y las leyes, así como mantener el sistema de inscripciones electorales y de registro de partidos políticos.

Con la información ya descargada del SERVEL se procede a separar los candidatos según tendencia política, sumar los datos y asignar los % de tendencias por comunas, asumiendo el 100% de los votantes.

3. Objetivos

3.1 Objetivos Generales

Montar una plataforma de Clúster, con comunicación eficiente, orientada al ámbito del procesamiento de un algoritmo en secuencial, el cual será paralelizado con el objetivo de simular la tendencia de las elecciones presidenciales 2017.

3.2 Objetivos Específicos

- Implementar un clúster de 5 nodos (1 maestro y 4 esclavos).
- Construir un ambiente secuencial y uno paralelo
- Obtener la información requerida por medio de un proceso paralelizable mejorando los tiempos de respuesta respecto a un modelo secuencial.
- Buscar el mejor rendimiento de los tiempos de respuesta de los procesos paralelos.
- Comparar el rendimiento de un proceso secuencial y uno paralelo.

4. Alcances

Los alcances de este proyecto se basan en la creación de un ambiente paralelo que satisfaga la lectura de una enorme cantidad de datos y que por medio de su procesamiento entregue un resultado final en el menos tiempo posible.

5. Planteamiento resolución del problema

Se busca una solución informática, en ambiente secuencial como en paralelo, la cual pueda realizar una simulación de la tendencia política del año 2017, contemplando el 100% del universo de los votantes chilenos en el año 2013.

Para la realización de este cometido, se pretendió el seguimiento de los siguientes pasos:

- Descargar la información de las votaciones de las comunas de Chile en el año 2013

  http://transparencia.servicioelectoral.cl/Resultados/2013.html
- Se separó los candidatos según su tendencia políticas.
- Se sumaron los datos y se asignó un % de tendencia por comuna.
- Creación de un algoritmo secuencial, que recibe los datos.
- Análisis de la estructura del algoritmo secuencial y determinar lo que se puede trabajar de manera paralela y lo que no.
- Creación de un código paralelo para recibir los datos.
- Análisis y optimización del código en paralelo.
- Análisis de rendimiento comparando la ejecución del código en paralelo y secuencial.

6. Marco teórico

6.1 Clúster

El término clúster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Hoy en día desempeñan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno. La tecnología de clústers ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software de misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clústers surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran. Simplemente, un clúster es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio. Los clústers son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

6.2 Nodos

Pueden ser simples ordenadores, sistemas multiprocesador o estaciones de trabajo (workstations). En informática, de forma muy general, un nodo es un punto de intersección o unión de varios elementos que confluyen en el mismo lugar. Ahora bien, dentro de la informática la palabra nodo puede referirse a conceptos diferentes según el ámbito en el que nos movamos:

- En redes de computadoras cada una de las máquinas es un nodo, y si la red es Internet, cada servidor constituye también un nodo.

- En estructuras de datos dinámicas un nodo es un registro que contiene un dato de interés y al menos un puntero para referenciar (apuntar) a otro nodo. Si la estructura tiene solo un puntero, la única estructura que se puede construir con él es una lista, si el nodo tiene más de un puntero ya se pueden construir estructuras más complejas como árboles o grafos.

El clúster puede estar conformado por nodos dedicados o por nodos no dedicados.

En un clúster con nodos dedicados, los nodos no disponen de teclado, ratón ni monitor y su uso está exclusivamente dedicado a realizar tareas relacionadas con el clúster. Mientras que, en un clúster con nodos no dedicados, los nodos disponen de teclado, ratón y monitor y su uso no está exclusivamente dedicado a realizar tareas relacionadas con el clúster, el clúster hace uso de los ciclos de reloj que el usuario del computador no está utilizando para realizar sus tareas.

Cabe aclarar que a la hora de diseñar un clúster, los nodos deben tener características similares, es decir, deben guardar cierta similaridad de arquitectura y sistemas operativos, ya que si se conforma un clúster con nodos totalmente heterogéneos (existe una diferencia grande entre capacidad de procesadores, memoria, disco duro) será ineficiente debido a que el middleware delegará o asignará todos los procesos al nodo de mayor capacidad de cómputo y solo distribuirá cuando este se encuentre saturado de procesos; por eso es recomendable construir un grupo de ordenadores lo más similares posible.

6.3 Clúster Beowulf

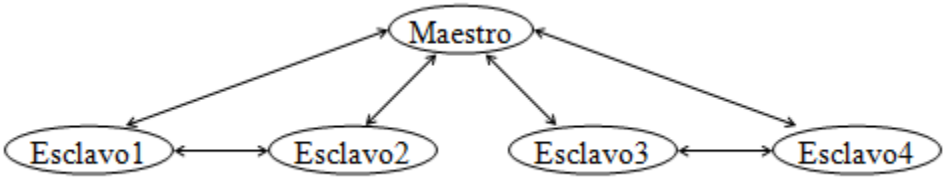
Para el desarrollo de la plataforma de clúster para el desarrollo del laboratorio, se montó un clúster de tipo Beowulf.

Un clúster Beorulf es una colección de ordenadores conectados en una red privada que pueden ser usados para computación paralela. Consisten en una comodidad de hadware utilizando software de uso libre, como Linux o BSD, usualmente junto con PVM (Parallel Virtual Machine) o MPI(Message Passing Interface). El estándar de conexión consiste en un un nodo maestro que controla un cierto número de nodos esclavos. Los nodos esclavos tienen un acceso general a todos los archivos del mismo servidos.

6.4 Open MPI

MPI significa "**M**essage **P**assing **I**nterface" y procede del esfuerzo de la comunidad de la computación paralela (industria y grupos de desarrollo de software libre) que creó el MPI Forum con el objetivo de desarrollar un estándar para programación paralela basada en paso de mensajes. El primer estándar se publicó en 1994 y, posteriormente, se generó una segunda versión "MPI-2" publicada en 1997. A partir de estas definiciones existen implementaciones tanto comerciales como de libre distribución (nosotros utilizaremos **MPICH**). MPI es, junto con **PVM** "**P**arallel **V**irtual **M**achine", uno de los paquetes de software más populares en el entorno de aplicaciones basadas en paso de mensajes.

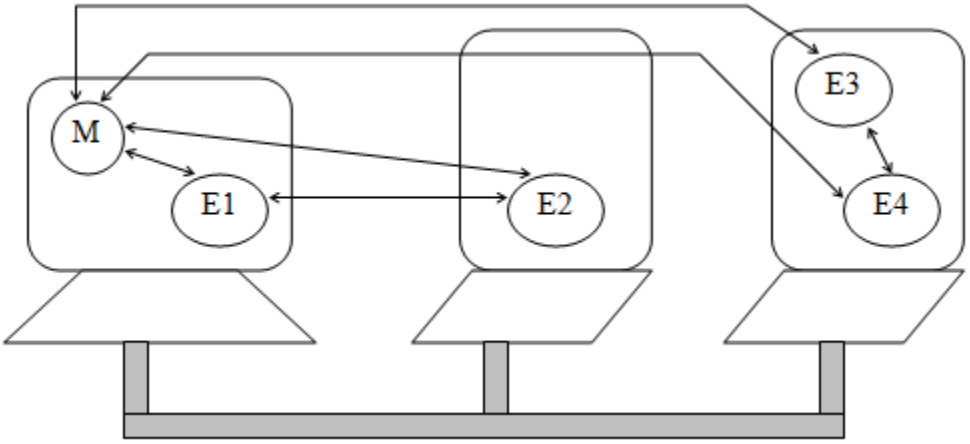
Una aplicación paralela típica sería aquella en la que un proceso maestro crea varios procesos esclavos para realizar, cada uno de ellos, una parte del trabajo global. Gráficamente, este modelo es el siguiente:



Donde el Maestro se comunica con sus esclavos, el Esclavo1 se comunica con el Esclavo2 y el Esclavo3 con el Esclavo4.

Con un soporte de sistema operativo multitarea (como UNIX), esta aplicación podría ejecutarse en una máquina monoprocesador multiplexando la CPU entre los cinco procesos que se crearían y comunicándose bien a través de memoria común o por un mecanismo de paso de mensajes.

Si disponemos de más de una máquina física (por ejemplo PC's o iMac's o Supercomputadores) que tienen interconectividad entre sí a través de una red de datos, MPI nos permitirá contemplar dichas máquinas físicas como un conjunto de nodos de computación (cluster) en el que ejecutar nuestra aplicación paralela de tal forma que los procesos se distribuyan entre las distintas CPU's disponibles y así poder ejecutarse con mayor paralelismo real. Supuesto que tenemos un PC y dos iMacs, la aplicación ejemplo podría ejecutarse tal y como se muestra a continuación:



Donde las comunicaciones entre los procesos que se ejecutan en máquinas distintas, fluirán a través de la red de datos.

Open MPI es quién lidera actualmente las implementaciones MPI-2, usadas por muchas supercomputadoras del TOP 500. Es de código abierto y desarrollado y mantenido por el consorcio académico, de investigación y la industria y asociados. El proyecto de Open MPI ha establecido como meta ser la mejor librería de interface para el traspaso de información.

6.5 Ruido

En comunicación, se denomina ruido a toda señal no deseada que se mezcla con la señal útil que se quiere transmitir. Es el resultado de diversos tipos de perturbaciones que tiende a enmascarar la información cuando se presenta en la banda de frecuencias del espectro de la señal, es decir, dentro de su ancho de banda.

Para medir la influencia del ruido sobre la señal se utiliza la relación señal/ruido, que generalmente se maneja en decibelios (dB). Como potencia de la señal se adopta generalmente la potencia de un tono de pruebas que se inyecta en el canal. La potencia del ruido suele medirse a la entrada del receptor, cuando por él no se emite dicho tono. Cuando se transmiten señales digitales por un canal, el efecto del ruido se pone de manifiesto en el número de errores que comete el receptor. Se deduce inmediatamente que dicho número es tanto mayor cuanto más grande sea la probabilidad de error.

La probabilidad de error depende del valor de la relación señal/ruido. Cuanto mayor sea esta relación, más destaca la señal sobre el ruido y, por tanto, menor es la probabilidad de error. Cuando el ruido se añade a una señal con distorsión, la probabilidad de error crece rápidamente.

La distorsión que produce el ruido en una determinada comunicación depende de su potencia, de su distribución espectral respecto al ancho de banda de la señal, y de la propia naturaleza información que transporta. El ruido afecta de diferente manera a la información que transportan las señales analógicas que a la codificada mediante señales digitales. Esta es la causa por la que se ha establecido una tipificación básica de los canales: los canales analógicos no es buena (con amplificación) y los canales digitales (con regeneración).

7. Diseño y arquitectura del software

Para la realización de este trabajo se montó un clúster de trabajo. Éste está enfocado en conseguir una gran capacidad de cálculo y realizar todas las operaciones necesarias para determinar la tendencia política de las siguientes elecciones.

Componentes del Clúster

A continuación se describen las características del ambiente en el cual se trabaja.

Nodo 1(Maestro)

- Procesador Intel Core I7

  Frecuencia 3,4 GHz

- Memoria Ram: 8GB, 7,88 GB disponibles

Nodo 2(Esclavo 1)

- Procesador Intel Core I7

  Frecuencia 3,4 GHz

- Memoria Ram: 8GB, 7,88 GB disponibles

Nodo 3(Esclavo 2)

- Procesador Intel Core I7

  Frecuencia 3,4 GHz

- Memoria Ram: 8GB, 7,88 GB disponibles

Nodo 4(Esclavo 3)

- Procesador Intel Core I7

  Frecuencia 3,4 GHz

- Memoria Ram: 8GB, 7,88 GB disponibles

Nodo 5(Esclavo 4)

- Procesador Intel Core I7

  Frecuencia 3,4 GHz

- Memoria Ram: 8GB, 7,88 GB disponibles

Se trabajó con el sistema operativo Linux distribución Ubuntu para desarrollar una plataforma para el procesamiento de información entregada por el SERVEL, y determinar la tendencia política para las próximas elecciones 2017.

Dentro de la distribución Ubuntu, se seleccionó su versión 64 bits, ya que se puede aprovechar en su totalidad la memoria RAM en esta versión, puesto que en la versión de 32 bits sólo se puede utilizar hasta un tamaño de 4 GB de RAM.

Conexiones de red:

- Red de interconexión:
- Topología:
- Velocidad:
- Equipamiento

8. Herramientas utilizadas

- Sistema Operativo: Linux, distribución 14.04. Permite el multiproceso y multiusario.
- Middleware: Open MPI. Api de código abierto, que facilita la programación paralela y permite la distribución de procesos de manera dinámica, logrando un alto redimiento y tolerancia a fallos.
- Protocolo de comunicación MPI: Es el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida.
- Sublime Text: editor de texto y de código, distribución gratuita.
- Htop: sistema interactivo de monitoreo de procesos escritos para Linux. Muestra los procesos que se están procesando en el computador y la cantidad de CPU usada.

9. Análisis de códigos

9.1 Código Secuencial

Luego de crear en primera instancia el código secuencial, se hizo una prueba con una cantidad pequeña de datos, de 200.000, proceso el cual tomó 10 minutos, leyendo archivos del tipo .xls. Primeramente parece un resultado prometedor, pero si se externaliza a la cantidad total de datos, que rodea los 12 millones, el proceso tomaría alrededor de 10 horas, asunto que no suena muy prometedor.

Para solucionar esta situación, se pasó a cambiar el tipo de archivo a uno de tipo csc, que disminuyó significativamente el tiempo del pequeño grupo de datos de 10 minutos a 4 segundos, que llevado a la cantidad total de datos tomaría alrededor de 40 segundos.

Debido a una limitación de la función csv.reader() fue necesario realizar un *split* a la información entregada por dicha función. Separando por ";". En función en que se iban encontrando los datos de una persona, se iba a buscar en la lista de comunas sus probabilidades pertenecientes a cada tendencia política. Una vez encontrada esta información, se generan los rangos de probabilidades para cada tendencia política, enviándolas como parámetros a la función Voto (rango1, rango2).

```
def contarVoto(comunas, inicio, fin):
    suma = 0
    sumb = 0
    sumc = 0
    for x in xrange(inicio, fin):
        archivo = 'Data/data_servel_' + str(x) + '.csv'
        print str(x)
        reader = csv.reader(open(archivo, 'rb'))
        for i, row in enumerate(reader):
            particion = str(row[0]).split(";")
            comuna = particion[1]
            sw = 0
            for j in range(len(comunas)):
                if comuna == comunas[j][0]:
                    rango1 = int(comunas[j][2])
                    rango2 = (int(comunas[j][2]) +
int(comunas[j][3]))
                    voto = Voto(rango1, rango2)
                    if voto == 1:
                        suma = suma + 1
                    elif voto == 2:
                        sumb = sumb + 1
                    else:
                        sumc = sumc + 1
                    sw = 1
            if sw == 0:
```

```
            print comuna + "Error"
    sumtotal = suma + sumb + sumc
    print " izquierda: " + str(suma)
    print " derecha: " + str(sumb)
    print " independiente: " + str(sumc)
    print " votos totales: " + str(sumtotal)
```

Impresión en pantalla de las tendencias políticas junto con el tiempo que tomó en ser ejecutado el algoritmo.

```
print "*****Secuencial*****"
tiempo_inicial = time()
comunas = LeerComunas()
contarVoto(comunas, 1, 14)
tiempo_final = time() - tiempo_inicial
print "tiempo total de ejecucion: " + str(tiempo_final)
```

9.2 Código en Paralelo

9.2.1 Granularidad Fina

Luego de revisar y probar el algoritmo paralelo de granularidad gruesa, se pensó en los beneficios que entregaría dividir los paquetes de datos, dejando el menor tamaño posible y así todos los procesadores trabajaran en paralelo.

Se pensaba de manera teórica, que al disminuir la cantidad de datos y aumentando la cantidad de procesadores se encontraría un mejor tiempo de procesamiento. Cabe señalar que el primer algoritmo secuencial se demoró aproximadamente 10 horas en procesar más de 12 millones de datos, con la primera optimización de cambio en el tipo de archivo se redujo a 15 minutos aproximadamente. Pasando al algoritmo paralelo esto este tiempo se redujo a 4,5 minutos de ejecución. Es por esto, que con la nueva teoría, se esperaba que el tiempo total de ejecución fuera menor a un minuto.

Para entender de mejor manera el código realizado, se describirá las funciones que representa ese trozo de código y luego se mostrara el código real. El código completo se adjuntara en un anexo.

Este código paralelo comienza llamando la función main(), la cual se encarga de verificar si está en el Rank maestro (Rank == 0) o en un Rank esclavo (Rank ¡=0). En caso de ser el Rank maestro, este enviara a cada Rank esclavo la información del número de Rank que fue activado, el listado de comunas y además el tiempo inicial de ejecución desde que se llama al Rank en una lista.

El listado de comunas contenida en el archivo “comunas.csv”, se envía a una lista para disminuir los tiempos de comparación en las próximas iteraciones.

```
def LeerComunas():
        # Apertura de archivo.
    reader = csv.reader(open('Data/comunas.csv', 'rb'))
    lista = []
    for i, 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
```

Para calcular de manera más precisa el tiempo de comunicación, se toma el tiempo actual y se guarda en la variable tiempo_inicial, luego el Rank “n” tomará el tiempo mencionado anteriormente y calculara una diferencia respecto al tiempo que existe al momento de llegar el paquete de datos al Rank “n”.

```
def main():
    if rank == 0:
        tiempo_inicial = time()
```

```
comunas = LeerComunas()

for i in range(1, 40):

    t_com_ini = time()

   listb = dict(numero=i, t_com_inicial=t_com_ini,
   comunas=comunas)

    comm.send(listb, dest=i)
```

En caso de que el Rank sea distinto a cero, quiere decir que se está en un esclavo “n”, se toma el tiempo actual para calcular el tiempo de comunicación y luego para calcular el tiempo total de procesamiento.

En este punto, fue necesario utilizar un *flag* para verificar el tiempo de comunicación y también para saber a qué Rank se le enviaban los datos.

```
print "comunicacion: " + str(t_comunicacion) + "- nodo: " +
str(lista["numero"])
```

Luego se guardó en una variable el dato “numero” perteneciente a la lista entregada por el Rank maestro. Con esto se eliminaron los errores de asignación y sintaxis que entregaba el lenguaje Python.

Finalmente el Rank “n” manda a llamar la función contarVoto(lista["comunas"], archivo, archivo + 1, rank, name, t_comunicacion, t_recibido) los parámetros enviados son:

- lista["comunas"] : contiene las probabilidades y tendencias políticas de cada comuna.
- archivo : Limite inferior de iteración que se deben realizar para abrir archivos que contienen los datos de las personas.
- archivo + 1 : Limite superior de las iteración que se deben realizar para abrir los archivos que contienen los datos de las personas. Como en esta granularidad cada procesador tomara un paquete de datos, por eso el límite superior es el límite inferior + 1.
- rank : Nodo que procesara los datos.
- name : Nombre del nodo.
- t_comunicacion : Tiempo de comunicación que luego se devolverá al nodo maestro.
- t_recibido : Tiempo inicial de procesamiento en el Rank “n”.

```
if rank != 0:
```

```
        t_recibido = time()
        lista = comm.recv(source=0)
        t_comunicacion = lista["t_com_inicial"] - t_recibido
        print "comunicacion: " + str(t_comunicacion) + "- nodo: "
+
       str(lista["numero"])
        archivo = lista["numero"]
        contarVoto(lista["comunas"], archivo, archivo + 1, rank,
name,
       t_comunicacion, t_recibido)
```

La función contarVoto(comunas, inicio, fin, rank, name, t_com, t_i) recibe los datos del rank. Esta función se pensó y creo de manera genérica, suponiendo que distintos factores harían disminuir la cantidad de procesadores manteniendo la cantidad de paquetes de datos. Es por esto que hay un método capaz de generar nuevos nombres para así guardar en una lista "n" cantidad paquetes de datos.

Debido a una limitación de la función csv.reader() fue necesario realizar un *split* a la información entregada por dicha función. Separando por ";"

Luego se procedió a buscar la información de la comuna que se estaba leyendo en ese momento, es decir, a medida que se encontraban datos de una persona se iba a buscar en la lista de comunas sus probabilidades pertenecientes a cada tendencia política. Una vez encontrada esta información, se generan los rangos de probabilidades para cada tendencia política, enviándolas como parámetros a la función Voto(rango1, rango2).

```
def contarVoto(comunas, inicio, fin, rank, name, t_com, t_i):
    suma = 0
    sumb = 0
    sumc = 0
    for x in xrange(inicio, fin):
        archivo = 'Data_40/data_servel_' + str(x) + '.csv'
```

```
        reader = csv.reader(open(archivo, 'rb'))
        for i, row in enumerate(reader):
            particion = str(row[0]).split(";")
            comuna = particion[1]
            for j in range(len(comunas)):
                if comuna == comunas[j][0]:
                    rango1 = int(comunas[j][2])
                    rango2 = (int(comunas[j][2]) +
int(comunas[j][3]))
                    voto = Voto(rango1, rango2)
```

La función Voto(rango1, rango2) se encarga de tomar los rangos y generar un numero aleatorio y asignarlo como un voto. Por ejemplo la comuna “lejos de aquí” tiene las siguientes probabilidades:

- Izquierda: 40%
- Derecha: 35%
- Independiente: 25%

Los rangos de esta comuna son:

- Rango 1 : [0,40]
- Rango 2 : [41,75]

Entonces, si se genera un numero aleatorio y este es pertenece al Rango 1, se asignara el voto a la tendencia de izquierda. En caso que el número pertenezca al Rango 2, se asignara el voto a la tendencia de derecha. Y por último caso, si el número aleatorio no pertenece a ninguno de los dos rangos, el voto se asignara a la tendencia independiente. Esta forma no presenta sesgos estadísticos.

```
def Voto(rango1, rango2):
    probabilidad = random.randint(1, 100)
    voto = 0
    if probabilidad <= rango1:
```

```
        voto = 1
    elif probabilidad > rango1 and probabilidad <= rango2:
        voto = 2
    else:
        voto = 3
    return voto
```

Teniendo los votos generados de la persona, se procede a sumar en sus respectivas tendencias políticas, quedando de la siguiente forma:

- suma = contiene todos los votos pertenecientes a la tendencia política de izquierda.
- sumb = contiene todos los votos pertenecientes a la tendencia política de derecha.
- sumc = contiene todos los votos pertenecientes a la tendencia política independiente.

Luego nuevamente se vuelve a sumar todas las tendencias para utilizar ese dato como *flag* y comprobar que se leyeron todos los datos. También se calculó el tiempo final de procesamiento, esto se realizó creando una variable con el tiempo actual y luego restar con la variable inicial que distribuyo el nodo maestro.

```
                    if voto == 1:
                        suma = suma + 1
                    elif voto == 2:
                        sumb = sumb + 1
                    else:
                        sumc = sumc + 1
    sumtotal = suma + sumb + sumc
    t_final = (time() - t_i)
    lista = dict(izquerda=suma, derecha=sumb, independiente=sumc,
                 total=sumtotal, t_com_nodo=t_com, tiempo=t_final)
    comm.send(lista, dest=0)
```

```
sys.stdout.write("Termino Proceso %d en %s.\n" % (rank, name))
```

Una vez que los rank esclavos terminan sus procesos, devuelven al rank maestro una lista con información relevante. Este rank comenzara a tomar esta información y hará un contador global de votos de cada tendencia política. Además con las informaciones de tiempos de comunicación se sumaran todos los valores devueltos por cada rank y con los tiempos de procesamiento de cada rank se eligira al mayor tiempo.

Finalmente se muestran estos datos por pantalla.

```
if rank == 0:
    izquerda = 0
    derecha = 0
    independiente = 0
    totales = 0
    t_mayor = -1
    t_com_total = 0
    for i in range(1, 40):
        voto = comm.recv(source=i)
        izquerda = izquerda + voto["izquerda"]
        derecha = derecha + voto["derecha"]
        independiente = independiente + voto["independiente"]
        totales = totales + voto["total"]
        t_com_total = t_com_total + voto["t_com_nodo"]
        if voto["tiempo"] > t_mayor:
            t_mayor = voto["tiempo"]
    print " izquierda: " + str(izquerda)
    print " derecha: " + str(derecha)
    print " independiente: " + str(independiente)
```

```
print " votos totales: " + str(totales)
print " Timpo comunicacion: " + str(t_com_total)
print " Tiempo Mayor de proceso: " + str(t_mayor)
tiempo_final = time() - tiempo_inicial
print "tiempo total de ejecucion: " + str(tiempo_final)
return 0
```

10. Análisis de Resultados

10.1 Comparación Secuencial v/s Paralelo

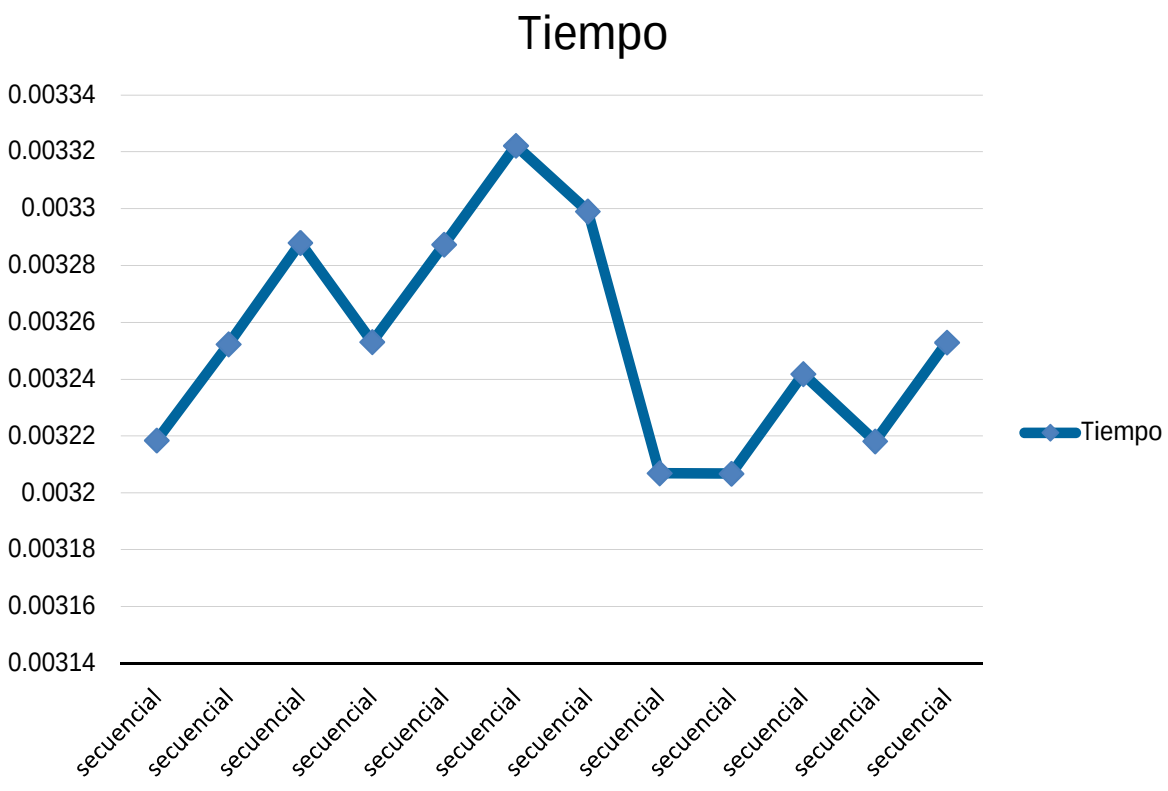
En esta sección se comparan los tiempos de ejecución del algoritmo secuencial y paralelo, con sus tiempos optimizados.

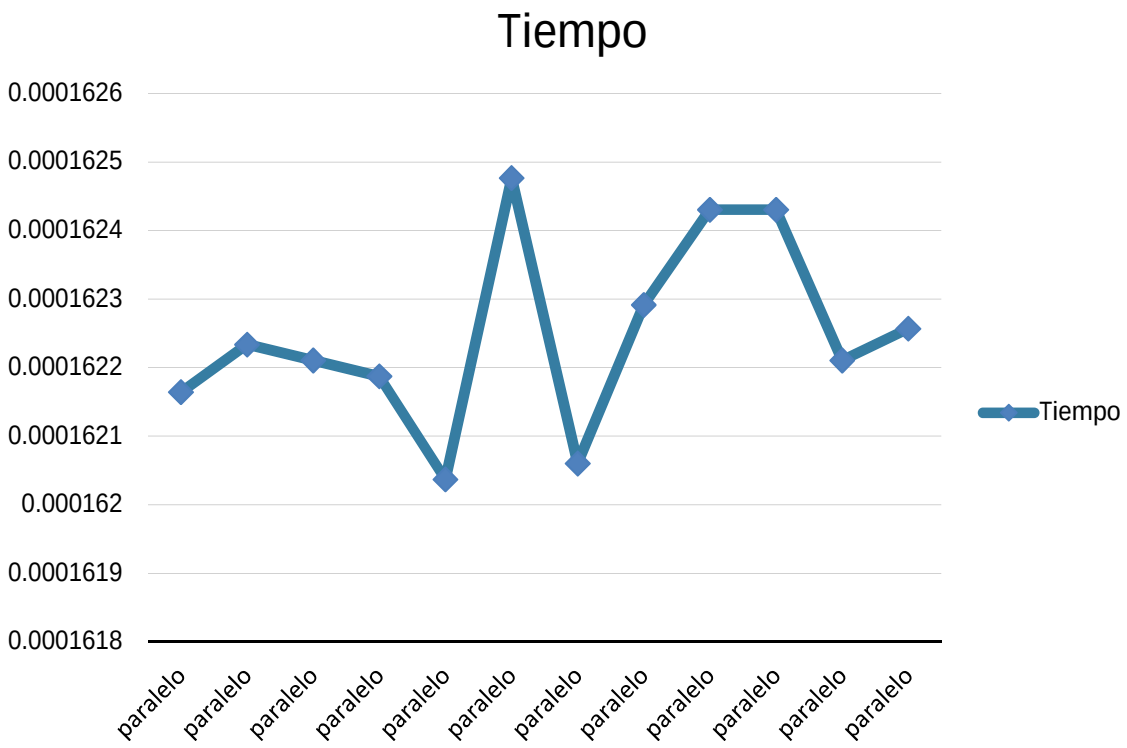
Descripción	Fecha	Tiempo	izquierda	derecha	independiente	% izquierda	% derecha	%independiente
Leer archivo, asignar voto y mostrar por pantalla	14 abr 2015	9m 55,085s						
Toma resultado todos los votos	14 abr 2015	9m 16,061s						
Cambio paso parámetros generar voto	15 abr 2015	9m 5,052s						
secuencial	21 abr 2015	4m 38,076s	5885203	3396709	3623975	45,601	26,319	28,080
secuencial	21 abr 2015	4m 41,001s	5884328	3397775	3623784	45,594	26,327	28,079
secuencial	21 abr 2015	4m 44,084s	5882940	3398086	3624861	45,583	26,330	28,087
secuencial	21 abr 2015	4m 41,070s	5882992	3397758	3625110	45,584	26,327	28,089
secuencial	21 abr 2015	4m 44,034s	5883154	3397701	3625032	45,585	26,327	28,088
secuencial	21 abr 2015	4m 47,039s	5885053	3396000	3624834	45,600	26,314	28,087
secuencial	21 abr 2015	4m 45,041s	5883098	3398069	3624720	45,585	26,330	28,086
secuencial	21 abr 2015	4m 37,079s	5884070	3395025	3626792	45,592	26,306	28,102
secuencial	21 abr 2015	4m 37,064s	5881589	3401876	3622422	45,573	26,359	28,068
secuencial	21 abr 2015	4m 40,097s	5883132	3398000	3624755	45,585	26,329	28,086
secuencial	21 abr 2015	4m 38,050s	5881235	3396481	3628171	45,570	26,317	28,113
secuencial	21 abr 2015	4m 41,054s	5882527	3395571	3627789	45,580	26,310	28,110
promedios		4m 41,141s				45,59	26,32	28,09

Tabla 1: Tiempos secuenciales

Descripción	Fecha	Tiempo	izquierda	derecha	independiente	% izquierda	% derecha	%independiente
paralelo	22 abr 2015	0m 14,011s	5883160	3397566	3625161	45,585	26,326	28,089
paralelo	22 abr 2015	0m 14,017s	5881338	3398776	3625773	45,571	26,335	28,094
paralelo	22 abr 2015	0m 14,015s	5885838	3392910	3627139	45,606	26,290	28,105
paralelo	22 abr 2015	0m 14,013s	5885563	3398057	3622267	45,604	26,330	28,067
paralelo	22 abr 2015	0m 14,000s	5883547	3399047	3623293	45,588	26,337	28,075
paralelo	22 abr 2015	0m 14,038s	5887225	3398368	3620294	45,617	26,332	28,051
paralelo	22 abr 2015	0m 14,002s	5885051	3395009	3625827	45,600	26,306	28,094
paralelo	22 abr 2015	0m 14,022s	5886689	3395671	3623527	45,612	26,311	28,077
paralelo	22 abr 2015	0m 14,034s	5882892	3398092	3624903	45,583	26,330	28,087
paralelo	22 abr 2015	0m 14,034s	5878851	3399849	3627187	45,552	26,343	28,105
paralelo	22 abr 2015	0m 14,015s	5883060	3397616	3625211	45,584	26,326	28,090
paralelo	22 abr 2015	0m 14,019s	5887150	3398393	3620344	45,616	26,332	28,052
promedios		0m 14,018s				45,59	26,32	28,08

Tabla 2: Tiempos paralelos


Tiempo
0.00334
0.00332
0.0033
0.00328
0.00326
0.00324
0.00322
0.0032
0.00318
0.00316
0.00314
secuencial
Tiempo


Tiempo
0.0001626
0.0001625
0.0001624
0.0001623
0.0001622
0.0001621
0.000162
0.0001619
0.0001618
paralelo
Tiempo

10.2 Análisis de tablas y gráficos de Algoritmo Secuencial vs Paralelo

Como se observa en cada gráfico de los tiempos para la plataforma en paralelo y la plataforma en secuencial, al aumentar la capacidad de cómputo, el tiempo que toma realizar los cálculos disminuye. Esta situación se debe a la gran cantidad de procesadores disponibles para la cantidad de 12.905.887 datos a trabajar.

11. Medidas de Rendimiento

Para las métricas de rendimiento, se calculó el Speed-Up, eficiencia y costos de tiempo para las ejecuciones en paralelo del algoritmo. Luego se analizaron los datos de cada métrica.

Prueba	Archivos	Tiempo
1	13	46,6011729
2	13	49,7198160
3	13	49,0631828
4	13	44,8300000
5	13	45,2362362
6	13	44,1214250
7	13	49,2145125
8	13	48,2515151
9	13	47,2323724
10	13	46,2362362
	Promedio	47,0506469

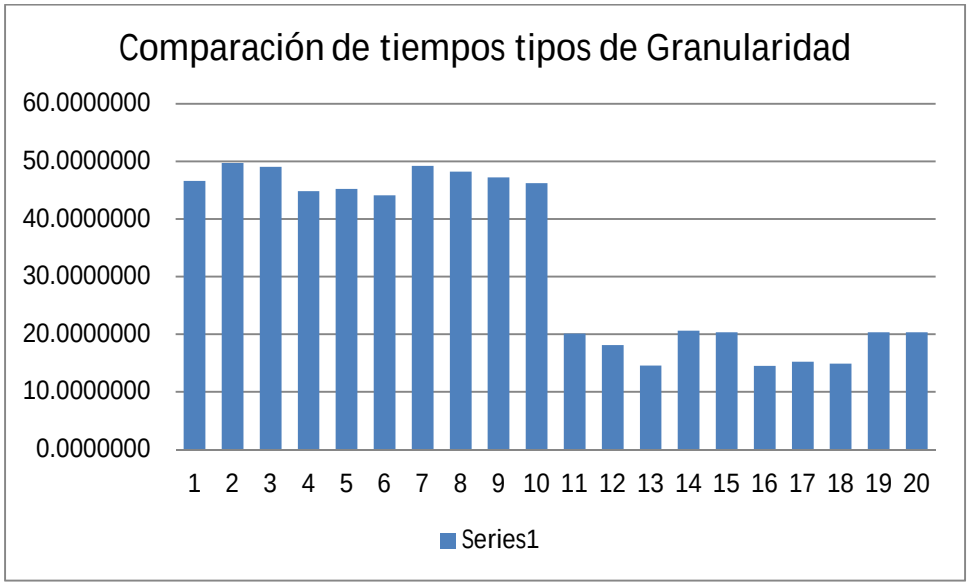
Tabla 3: Tiempos de ejecución granularidad gruesa

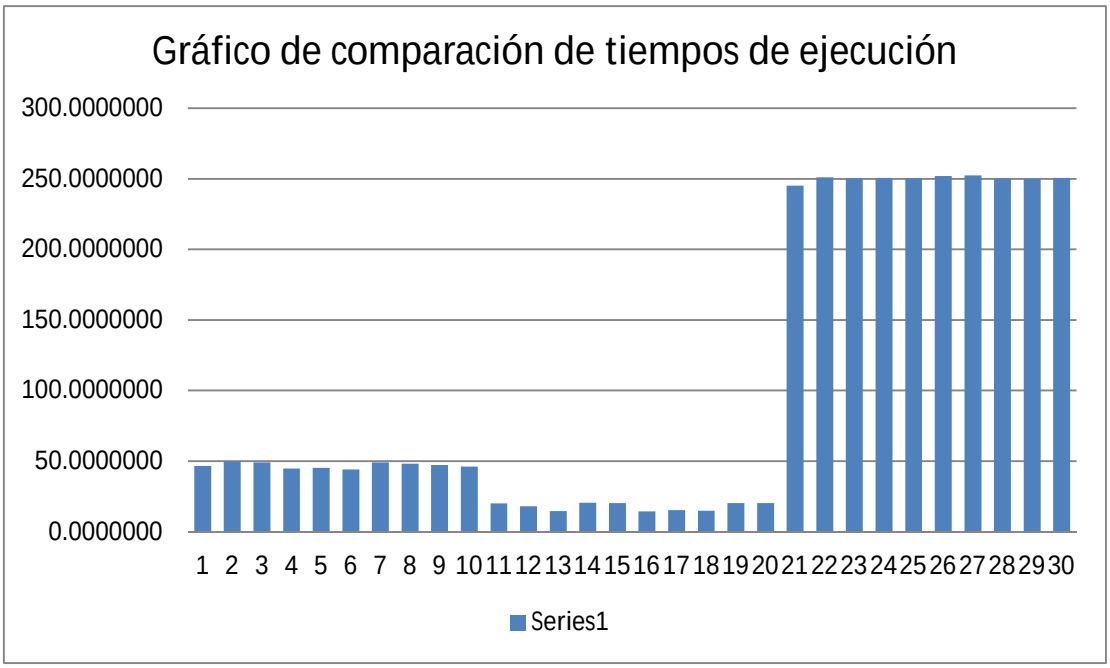
Prueba	Archivos	Tiempo
1	13	20,1251500
2	13	18,1251520
3	13	14,5732152
4	13	20,6233600
5	13	20,3553000
6	13	14,5342152
7	13	15,2342152
8	13	14,9003252
9	13	20,3255150
10	13	20,3329990
	Promedio	17,9129447

Tabla 4: Tiempos de ejecución granularidad fina

Prueba	Archivos	Tiempo
1	13	245,2323320
2	13	250,9942520
3	13	250,3235252
4	13	250,6223433
5	13	250,3636323
6	13	251,9353532
7	13	252,3223332
8	13	250,2362362
9	13	250,0009330
10	13	250,7232350
	Promedio	250,2754175

Tabla 5: Tiempos de ejecución algoritmo secuencial





Como se puede observar en los gráficos expuestos y existe una gran variación del tiempo al momento de procesar la información aplicando una solución en un ambiente secuencial y en un ambiente paralelo aplicando granularidad gruesa y granularidad fina.

11.1 Speed-Up

El Speed-Up se define como el tiempo de ejecución serie del mejor algoritmo para resolver el problema dado, dividido por el tiempo utilizado por el sistema paralelo con p procesadores idénticos e iguales al utilizado en el entorno secuencial.

- Peor tiempo secuencial: 250,994252
- Mejor tiempo paralelo granularidad gruesa: 44,121425
- Peor tiempo paralelo granularidad gruesa: 49,7198160
- Mejor tiempo paralelo granularidad fina: 14,5342152

$$SpeedUp_{G.Gruesa} = \frac{Peor\ tiempo\ secuencial}{Mejor\ tiempo\ paralelo\ granularidad\ gruesa} = \frac{250{,}994252}{44{,}121425}$$

$$SpeedUP_{G.Gruesa} = 5.6887159016282906$$

$$SpeedUp_{G.Fina} = \frac{Peor\ tiempo\ granularidad\ gruesa}{Mejor\ tiempo\ paralelo\ granularidad\ fina} = \frac{49{,}7198160}{14{,}5342152}$$

$$SpeedUp_{G.Fina} = 3.4208806816070812$$

Como se puede observar, el Speed-Up de la granularidad gruesa con respecto al algoritmo secuencial, presenta una mejor de un 568,87%, lo que denota una mejora bastante considerable, ya que llega a ser una mejora de casi 6 veces más.

Para el Speed-Up de la granularidad fina con respecto a la granularidad gruesa, se observar una mejora de un 342,08%, lo que demuestra que el algoritmo en paralelo podía presentar aún mayores mejoras por el modo en que se tratan los datos.

11.2 Eficiencia

La eficiencia es una medida de tiempo que un procesador emplea de forma útil. Se define como el coeficiente entre el Speed-Up y el número de procesadores, lo cual da una relación de eficiencia de ejecución con respecto a los procesadores utilizados por esta.

Para el caso de la eficiencia, se medirá con respecto a la cantidad total de procesadores, para observar la capacidad máxima que se cuenta para la realización de este proyecto, y porque al momento de disminuir la cantidad de procesadores para el manejo de los datos, se observó que el aumento del tiempo era proporcional.

$$Eficiencia_{G.Gruesa} = \frac{SpeedUp_{G.Gruesa}}{p} = \frac{5.6887159016282906}{40}$$

$$Eficiencia_{G.Gruesa} = 0.142217897540707265$$

$$Eficiencia_{G.Fina} = \frac{SpeedUp_{G.Fina}}{p} = \frac{3.4208806816070812}{40}$$

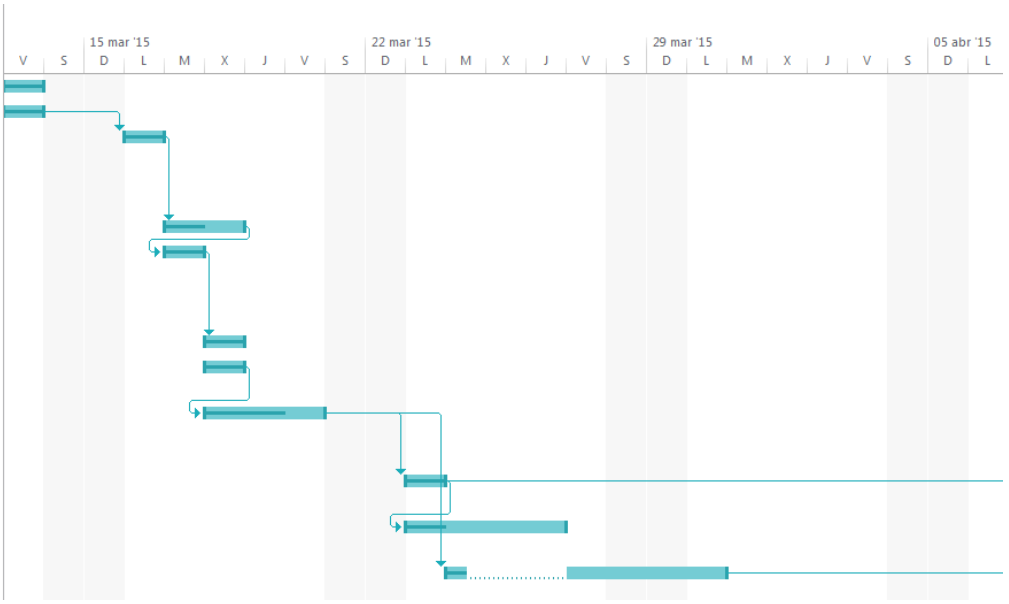
$$Eficiencia_{G.Fina} = 0.08552201704017703$$

Al hacer la comparación de eficiencia de granularidad gruesa y granularidad fina, se puede denotar que el algoritmo de granularidad gruesa es mucho más eficiente, ya que aprovecha de mejor manera los recursos.

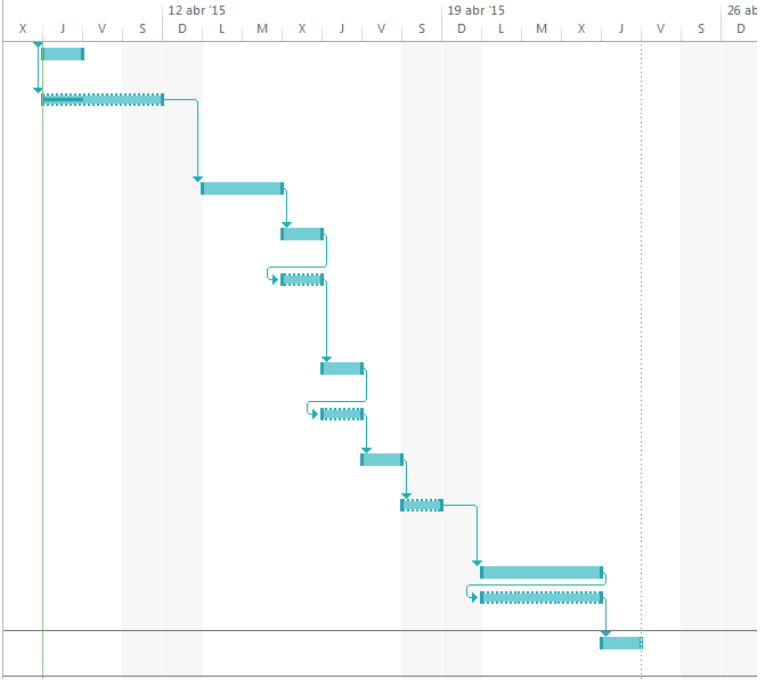
12. Carta Gantt

A continuación se detalla la planificación del proyecto con sus tareas y duraciones de éstas.

	Cód esc	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
1			Obtencion de datos	1 día	vie 13-03-15	vie 13-03-15
2			Fijacion de objetivos	1 día	vie 13-03-15	vie 13-03-15
3			Confeccion documento propuesta de proyecto	1 día	lun 16-03-15	lun 16-03-15
4			Entrega propuesta	2 días	mar 17-03-15	mié 18-03-15
5			Confeccion documento propuesta con correcciones	1 día	mar 17-03-15	mar 17-03-15
6			Entrega propuesta	1 día	mié 18-03-15	mié 18-03-15
7			Analisis de recursos informaticos	1 día	mié 18-03-15	mié 18-03-15
8			Modificaciones a recursos informaticos	3 días	mié 18-03-15	vie 20-03-15
9			Generar algoritmo secuencial	1 día	lun 23-03-15	lun 23-03-15
10			Programacion de algoritmo secuencial	4 días	lun 23-03-15	jue 26-03-15
11			Crear cluster de 4 nodos	2,5 días	mar 24-03-15	lun 30-03-15



	Cód de	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
12			Entrega avance 1	1 día	jue 09-04-15	jue 09-04-15
13			Analisis del algoritmo secuencial que debe ser paralelizado	3 días	jue 09-04-15	sáb 11-04-15
14			Construccion algoritmo paralelo	2 días	lun 13-04-15	mar 14-04-15
15			Optimizacion codigo paralelo	1 día	mié 15-04-15	mié 15-04-15
16			Analizar rendimiento al aumentar la cantidad de nodos	1 día	mié 15-04-15	mié 15-04-15
17			Aplicar granularidad gruesa y fina	1 día	jue 16-04-15	jue 16-04-15
18			Comparacion de granularidad	1 día	jue 16-04-15	jue 16-04-15
19			Testing y registro de resultados	1 día	vie 17-04-15	vie 17-04-15
20			Comparacion de paralelo versus secuencial	1 día	sáb 18-04-15	sáb 18-04-15
21			Revision de errores	3 días	lun 20-04-15	mié 22-04-15
22			Confeccion Informe final	3 días	lun 20-04-15	mié 22-04-15
23			Entrega Informe final	1 día	jue 23-04-15	jue 23-04-15



13. Conclusiones

Gracias a que el grupo de trabajo se mantuvo organizado, se pudo llevar a cabo los objetivos descritos al principio del informe.

En síntesis, se pudo montar una plataforma paralela a raíz de un algoritmo secuencial, pudiendo mejorar los tiempos significativamente, ya que en primera instancia se obtuvo un tiempo 6000 segundos aproximadamente de manera secuencial, llegando a un tiempo de máximo 14 segundos haciendo el mismo proceso en un ambiente paralelo.

Gracias a esto, pudimos darnos cuenta que la plataforma paralela es muy eficiente a la hora de manjar una gran cantidad de datos, algo que solamente estaba como hipótesis para la resolución del problema planteado y que se pudo comprobar al momento de llevarlo a la práctica. Pero para poder realizar una plataforma paralela, se requiere mucho más recursos que un ambiente secuencial, por eso, a la hora de plantear si se realiza una solución de manera paralela o secuencial, se debe evaluar las características de lo que se pide realizar y los alcances de los requerimientos.

También se concluye que el ruido del switch, que se debe a múltiples causas: a los componentes electrónicos (amplificadores), al ruido térmico de los resistores, a las interferencias de señales externas, etc. , varió por la demanda de ancho de banda, por ejemplo: como la red de la universidad es compartida, cuando nos encontrábamos temprano en el laboratorio se observó una demora de 35 segundos y al momento de llegar más alumnos al laboratorio a usar los computadores y su red, el tiempo aumentó a 40 segundos y luego a 42.Es imposible eliminar totalmente el ruido, ya que los componentes electrónicos no son perfectos. Sin embargo, es posible limitar su valor de manera que la calidad de la comunicación resulte aceptable.

14. Bibliografía

http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/

http://publiespe.espe.edu.ec/articulos/sistemas/arquitectura/arquitectura.htm

http://www.tomechangosubanana.com/tesis/escrito-1-split/node1.html

https://bitbucket.org/mpi4py/mpi4py

http://en.wikibooks.org/wiki/Building_a_Beowulf_Cluster

http://fscked.org/writings/clusters/cluster-1.html

https://www.linux.com/community/blogs/133-general-linux/9401

http://www.aosabook.org/en/openmpi.html

http://www.open-mpi.org/

15. Anexos

15.1 Anexo A: Implementación del clúster.

Para la implementación del clúster, se siguió una guía vista en una ayudantía anterior de Comunicación de Datos.

A modo ejemplo:

Nodo Maestro.		
Nombre de usuario: **cluster**		nombre de carpeta maestra**: mpi**
Ip maestro**: 10.1.10.113**	ip esclavo 1**: 10.1.10.114**	ip esclavo 2**: 10.1.10.115**

Paso 1: actualización de SO e instalación de programas a utilizar.

```
sudo apt-get update sudo apt-get upgrade
sudo apt-get install gcc g++
sudo apt-get install openmpi-bin openmpi-common libopenmpi1.3
libopenmpi-dev
sudo apt-get install ssh
sudo apt-get install nfs-kernel-server nfs-common portmap sudo
apt-get install build-essential
```

Paso 2: Creación de carpeta a compartir.

- *Posicionarse en carpeta* **cluster** cd /home/**cluster**
mkdir .ssh
sudo chmod 777 .ssh
cd ../../
sudo mkdir /**mpi** sudo chmod 777 /**mpi**
cd /**mpi**
sudo chown **user** /**mpi**

Paso 3: Editar ficheros hosts y exports

sudo gedit /etc/hosts

```
127.0.0.1  localhost

#     "ip"    "usuario"  "equipo"
      10.1.10.113  cluster        n13        # nodo maestro
      10.1.10.014  cluster        n14        # nodo esclavo 1
      10.1.10.015  cluster        n15        # nodo esclavo 2


# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
```

sudo gedit /etc/exports

```
# /etc/exports: the access control list for filesystems which may
be exported
# to NFS clients. See exports(5).

gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)

/mpi *(rw,sync)
```

Paso 4: Compartición de directorio con nodos esclavo.
sudo service nfs-kernel-server restart
- ESPERAR A QUE LOS ESCLAVOS MONTEN LA CARPETA
**** COMPLETAR CONFIGURACION NODO ESCLAVO*

Paso 5: Configuración SSH
ssh-keygen -t rsa
**** DARLE OK/ENTER A TODO*
*-posicionarse en carpeta .ssh cd /home/***cluster***/.ssh*
cat id_rsa.pub >> authorized_keys ssh-copy-id **10.1.10.114** ssh-copy-id **10.1.10.115**
- *Confirmar si el proceso es correcto* ssh **10.1.10.114**
**Al ingresar al nodo esclavo este no debe pedir contraseña.*

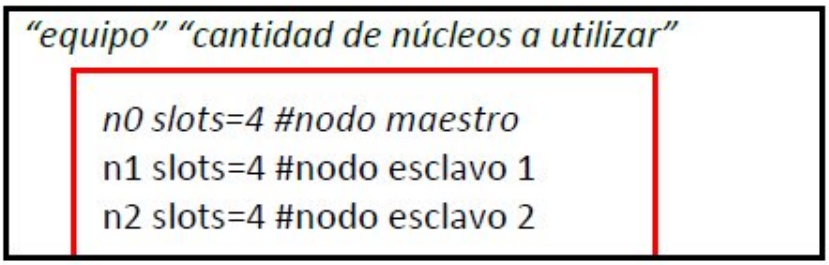
Paso 6: Ejecutar Código.

(ingresar a carpeta maestra)
*cd /home/cluster/**mpi***

compilar
mpicc –o **programa programa.c**

- *Ejecutar programa*
mpirun –np 4 –-host **10.1.10.114** ./**programa**

sudo gedit /home/hostfile



mpirun –np –-hostfile **/home/hostfile** ./**programa**

Nodo Esclavo

Paso 1: Actualización de SO e instalación de programas a utilizar.

```
sudo apt-get update sudo apt-get upgrade
sudo apt-get install gcc g++
sudo apt-get install openmpi-bin openmpi-common libopenmpi1.3
libopenmpi-dev
sudo apt-get install ssh
sudo apt-get install nfs-kernel-server nfs-common portmap
```

Paso 2: Creación de carpeta esclavo a sincronizar.
- *Ubicarse en la carpeta personal.* cd /home/***cluster***

- *Crear carpeta .shh y darle los permisos.* mkdir .ssh

sudo chmod 700 .ssh
- *Volver al directorio raíz (desde la carpeta personal)* cd ../../

- *Crear carpeta a compartir y otorgarle todos los permisos* sudo mkdir /**mpi**
sudo chmod 777 /**mpi**
- *Ingresar a la carpeta y cambiar el propietario del directorio* cd /**mpi**

sudo chown **cluster** /**mpi**

Paso 3: Montaje de carpeta maestra.
- Montar carpeta maestra compartida

- En cada nodo cliente ejecutar:

sudo mount **10.1.10.113**:/**mpi** /**mpi**
sudo gedit /etc/fstab

```
10.1.10.113:/mpi /mpi nfs
```

```
sudo mount –a
```

15.2 Anexo B: Pruebas de clúster.

Para realizar las pruebas de clúster, se utilizó el código de "**hola_mundo_mpi.py**", usando 40 procesos para poder usar la máxima cantidad de procesadores que tenemos disponibles.

```
from mpi4py import MPI
import sys
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()
sys.stdout.write(
 "Hello, World! I am process %d of %d on
%s.\n"
 % (rank, size, name))
```

Al momento de hacer esta prueba, el nodo maestro entregó la información cuántos procesadores estaban haciendo el proceso y qué nodo lo estaba ejecutando.

Rank: Procesador.

Size: Cantidad de procesos.

Name: Nombre del equipo en que se está ejecutando.

La línea de comando para poder ejecutar un proceso en paralelo es:

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

15.3 Anexo C: Cálculo de esfuerzo

					Recurso 1			Recurso 2		
Tarea	**Duración (Diás)**	**Fecha Inicio**	**Fecha Fin**	**Horas Actividad**	**% Dedicación**	**Rol**	**Costo UF**	**% Dedicación**	**Rol**	**Costo UF**
Entrega anteproyecto	1	24-03-2015	24-03-2015	1	100%	ingeniero	1,80	100%	ingeniero	1,80
Revisión anteproyecto	1	07-04-2015	07-04-2015	1	100%	ingeniero	1,80	100%	ingeniero	1,80
Analisis recursos necesarios	1	10-04-2015	10-04-2015	3	20%	ingeniero	1,08	20%	jefe proyecto	0,90
Instalación software necesario	5	13-04-2015	17-04-2015	25	0%	ingeniero	0,00	0%	jefe proyecto	0,00
Creación cluster	4	20-04-2015	23-04-2015	20	100%	tecnico	2,40	100%	tecnico	2,40
Modelacion base de datos	1	24-04-2015	24-04-2015	4	0%	ingeniero	0,00	100%	modelador	5,60
Diseño de sitio web	4	27-04-2015	30-04-2015	20	0%	desarrollador	0,00	0%	desarrollador	0,00
revision etapa 1	2	04-05-2015	05-05-2015	10	100%	ingeniero	18,00	100%	jefe proyecto	15,00
correcciones	3	06-05-2015	08-05-2015	15	30%	ingeniero	8,10	20%	jefe proyecto	4,50
Llenar base de datos	2	12-05-2015	13-05-2015	10	0%	ingeniero	0,00	20%	jefe proyecto	3,00
Creación página web	7	12-05-2015	20-05-2015	40	40%	ingeniero	28,80	0%	jefe proyecto	0,00
Montar página web en cluster	4	22-05-2015	27-05-2015	15	0%	ingeniero	0,00	0%	jefe proyecto	0,00
revision etapa 2	2	28-05-2015	29-05-2015	10	100%	ingeniero	18,00	100%	jefe proyecto	15,00
correcciones	3	01-06-2015	03-06-2015	15	30%	ingeniero	8,10	20%	jefe proyecto	4,50
Pruebas de rendimiento	4	04-06-2015	10-06-2015	20	100%	analista	14,00	100%	analista	14,00
Mejoras al sistema	5	11-06-2015	17-06-2015	25	100%	ingeniero	45,00	100%	programador	12,50
Pruebas de rendimiento y desempeño	7	18-06-2015	26-06-2015	40	50%	ingeniero	36,00	50%	jefe proyecto	28,00
revision final	2	29-06-2015	30-06-2015	15	100%	ingeniero	27,00	100%	jefe proyecto	22,50
Totales	**58**			**274,00**			**183,08**			**109,00**

					Recurso 3			Recurso 4		
Tarea	**Duración (Diás)**	**Fecha Inicio**	**Fecha Fin**	**Horas Actividad**	**% Dedicación**	**Rol**	**Costo UF**	**% Dedicación**	**Rol**	**Costo UF**
Entrega anteproyecto	1	24-03-2015	24-03-2015	1	100%	ingeniero	1,80	100%	ingeniero	1,80
Revisión anteproyecto	1	07-04-2015	07-04-2015	1	100%	ingeniero	1,80	100%	ingeniero	1,80
Analisis recursos necesarios	1	10-04-2015	10-04-2015	3	100%	analista	2,10	100%	analista	2,10
Instalación software necesario	5	13-04-2015	17-04-2015	25	0%	analista	0,00	100%	tecnico	3,00
Creación cluster	4	20-04-2015	23-04-2015	20	60%	analista	8,40	0%	tecnico	0,00
Modelacion base de datos	1	24-04-2015	24-04-2015	4	0%	analista	0,00	0%	tecnico	0,00
Diseño de sitio web	4	27-04-2015	30-04-2015	20	100%	desarrollador	2,80	100%	desarrollador	2,80
revision etapa 1	2	04-05-2015	05-05-2015	10	0%	analista	0,00	0%	programador	0,00
correcciones	3	06-05-2015	08-05-2015	15	100%	programador	7,50	100%	programador	7,50
Llenar base de datos	2	12-05-2015	13-05-2015	10	100%	modelador	14,00	0%	programador	0,00
Creación página web	7	12-05-2015	20-05-2015	40	100%	desarrollador	0,56	100%	desarrollador	0,56
Montar página web en cluster	4	22-05-2015	27-05-2015	15	0%	desarrollador	0,00	100%	tecnico	1,80
revision etapa 2	2	28-05-2015	29-05-2015	10	0%	analista	0,00	0%	programador	0,00
correcciones	3	01-06-2015	03-06-2015	15	100%	programador	7,50	100%	programador	7,50
Pruebas de rendimiento	4	04-06-2015	10-06-2015	20	100%	analista	14,00	100%	analista	14,00
Mejoras al sistema	5	11-06-2015	17-06-2015	25	100%	programador	12,50	100%	programador	12,50
Pruebas de rendimiento y desempeño	7	18-06-2015	26-06-2015	40	100%	analista	28,00	100%	analista	28,00
revision final	2	29-06-2015	30-06-2015	15	0%	analista	0,00	0%	programador	0,00
Totales	**58**			**274,00**			**100,96**			**83,36**

					Recurso 5			
Tarea	**Duración (Diás)**	**Fecha Inicio**	**Fecha Fin**	**Horas Actividad**	**% Dedicación**	**Rol**	**Costo UF**	**Costo total UF**
Entrega anteproyecto	1	24-03-2015	24-03-2015	1	100%	ingeniero	1,80	9,00
Revisión anteproyecto	1	07-04-2015	07-04-2015	1	100%	ingeniero	1,80	9,00
Analisis recursos necesarios	1	10-04-2015	10-04-2015	3	60%	tecnico	0,22	6,40
Instalación software necesario	5	13-04-2015	17-04-2015	25	100%	tecnico	3,00	6,00
Creación cluster	4	20-04-2015	23-04-2015	20	0%	tecnico	0,00	13,20
Modelacion base de datos	1	24-04-2015	24-04-2015	4	100%	modelador	5,60	11,20
Diseño de sitio web	4	27-04-2015	30-04-2015	20	100%	desarrollador	2,80	8,40
revision etapa 1	2	04-05-2015	05-05-2015	10	0%	desarrollador	0,00	33,00
correcciones	3	06-05-2015	08-05-2015	15	100%	tecnico	1,80	29,40
Llenar base de datos	2	12-05-2015	13-05-2015	10	0%	analista	0,00	17,00
Creación página web	7	12-05-2015	20-05-2015	40	100%	desarrollador	0,56	30,48
Montar página web en cluster	4	22-05-2015	27-05-2015	15	100%	tecnico	1,80	3,60
revision etapa 2	2	28-05-2015	29-05-2015	10	0%	desarrollador	0,00	33,00
correcciones	3	01-06-2015	03-06-2015	15	100%	tecnico	1,80	29,40
Pruebas de rendimiento	4	04-06-2015	10-06-2015	20	100%	analista	14,00	70,00
Mejoras al sistema	5	11-06-2015	17-06-2015	25	100%	programador	12,50	95,00
Pruebas de rendimiento y desempeño	7	18-06-2015	26-06-2015	40	100%	analista	28,00	148,00
revision final	2	29-06-2015	30-06-2015	15	0%	desarrollador	0,00	49,50
Totales	**58**			**274,00**			**75,68**	**552,08**

15.4 Anexo D: Cálculo del tiempo

Para calcular el tiempo que tomaban los algoritmos secuencial y paralelo, se agregó una función simple para el cálculo del tiempo, y que nos permitió tomar las métricas necesarias para el desarrollo de la actividad.

```
from time import time, strftime, localtime


def test():
    for i in range(100000):
        "Hello, world!".replace("Hello", "Goodbye")

start_time = time()
test()
elapsed_time = time() - start_time
print("Tiempo Transcurrido: %.10f seconds." % elapsed_time)
print strftime("%a, %d %b %Y %H:%M:%S", localtime())
```

15.5 Anexo E: Ejemplo de código paralelo

En la creación del código paralelo para el desarrollo del laboratorio, se utilizó un código en paralelo de ejemplo, facilitado por compañeros de carrera que ya cursaron la asignatura Computación Paralela.

```
from mpi4py import MPI
import sys

size1 = MPI.COMM_WORLD.Get_size()
rank1 = MPI.COMM_WORLD.Get_rank()
name1 = MPI.Get_processor_name()

comm = MPI.COMM_WORLD  # comunicador entre dos procesadores

rank = comm.rank     # id procesador actual
size = comm.size     #
process = MPI.Get_processor_name()

if rank == 0:
    lista_enviar = [1, 2, 3]
    comm.send(lista_enviar, dest=1)
    comm.send(lista_enviar, dest=2)
    sys.stdout.write(
        "Hello, World! I am process %d of %d on %s.\n"
        % (rank1, size, process))
#    comm.send (lista_enviar, dest = 9)

if rank == 1:
    lista_recibir = comm.recv(source=0)
    lista_recibir.append(10)  # agregar al final #
    comm.send(lista_recibir, dest=0)
    sys.stdout.write(
        "Hello, World! I am process %d of %d on %s.\n"
        % (rank, size, process))

if rank == 2:
    lista_recibir = comm.recv(source=0)
    lista_recibir.append(7)  # agregar al final #
    comm.send(lista_recibir, dest=0)
    sys.stdout.write(
        "Hello, World! I am process %d of %d on %s.\n"
        % (rank, size, process))

if rank == 3:
    lista_recibir = comm.recv(source=0)
    lista_recibir.append(8)  # agregar al final #
    comm.send(lista_recibir, dest=0)
    sys.stdout.write(
        "Hello, World! I am process %d of %d on %s.\n"
        % (rank, size, process))

if rank == 0:
    lista_uno = comm.recv(source=1)
    lista_dos = comm.recv(source=2)
    sys.stdout.write(
        "Hello, World! I am process %d of %d on %s.\n"
        % (rank1, size1, name1))
    print "He recibido el ", lista_uno + lista_dos
```

15.6 Código en Paralelo Granularidad Gruesa

Luego de proceso de código secuencial se prosiguió a hacer el código en paralelo. Para esto utilizamos la misma cantidad de procesadores que como archivos, en este caso tenemos 13 archivos con 1 millos de datos cada uno aproximadamente, es decir, utilizamos 13 procesadores para poder llegar al máximo de procesamiento en paralelo.

El código hace lo mismo que el secuencial pero va a asignándole tareas a través de un ciclo FOR a los demás nodos. Además el código tiene en promedio 47,0506469[seg] de ejecución.

A continuación se presenta el código:

```
from mpi4py import MPI
import csv
import random
import sys
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
```

```
def LeerComunas():
    # Apertura de archivo.
    reader = csv.reader(open('Data/comunas.csv', 'rb'))
    lista = []
    for i, 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
# la funcion Voto. Teniendo el voto generado, se asigna al
# partido politico.
def contarVoto(comunas, inicio, fin, rank, name):
    wt = MPI.Wtime()
    suma = 0
    sumb = 0
    sumc = 0
    for x in xrange(inicio, fin):
        # generacion del nombre "archivo" para leer la cantidad
necesaria de
        # archivos.
        archivo = 'Data/data_servel_' + str(x) + '.csv'
        print "Archivo: " + str(x)
        # Guardar la informacion contenida en el archivo CSV para
        # disminuir el tiempo de lectura.
        reader = csv.reader(open(archivo, 'rb'))
        for i, row in enumerate(reader):
            # Particionar la informacion con el delimitador ";"
            particion = str(row[0]).split(";")
            comuna = particion[1]
            sw = 0
            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
                    sw = 1
            if sw == 0:
                print comuna + "hola"
    sumtotal = suma + sumb + sumc
    wt = MPI.Wtime() - wt
    lista = dict(
        izquerda=suma, derecha=sumb, independiente=sumc,
total=sumtotal)
    comm.send(lista, dest=0)
    sys.stdout.write("Termino Proceso %d en %s.\n" % (rank, name))
def main():
    if rank == 0:
        print "*****Paralelo Granularidad Gruesa*****"
        tiempo_inicial = time()
        if (size > 13):
            for i in xrange(1, 14):
                comm.send(i, dest=i)
            for i in xrange(14, size):
                comm.send(0, dest=i)
        else:
            for i in xrange(1, size):
                comm.send(i, dest=i)
    if rank != 0:
        archivo = comm.recv(source=0)
        if archivo == 0:
            lista = dict(
                izquerda=0, derecha=0, independiente=0, total=0,
time=0)
            comm.send(lista, dest=0)
            sys.stdout.write("NO hay mas Archivos, Termino Proceso
```

```
%d en %s.\n" % (rank, name))
        else:
            comunas = LeerComunas()
            contarVoto(comunas, archivo, archivo + 1, rank, name)
    if rank == 0:
        izquerda = 0
        derecha = 0
        independiente = 0
        totales = 0
        for i 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"]
        print " Izquierda: " + str(izquerda)
        print " Derecha: " + str(derecha)
        print " Independiente: " + str(independiente)
        print " Votos totales: " + str(totales)
        tiempo_final = time() - tiempo_inicial
        print "Tiempo total de ejecucion: " + str(tiempo_final)
        print "Programa Finalizado"
        return 0
main()
```