

# Universidad Peruana de Ciencias Aplicadas



## TRABAJO PARCIAL

Programación Concurrente y Distribuida

Carrera de Ciencias de la Computación

Sección: CC72

Código	Nombres y apellidos
U201922773	Alejandro Olaf López Flores
U20201B122	Gleider Venancio Castro Ataucusi

Abril 2024

# Índice

<b>1. Introducción.....</b>	<b>3</b>
<b>2. Algoritmo Secuencial(K-Means) secuencial y de manera concurrente.....</b>	<b>3</b>
2.1. Explicación del Algoritmo Secuencial(K-Means).....	3
2.1.1. Generación de puntos aleatorios:.....	3
2.1.2. Inicialización de Clusters:.....	3
2.1.3. Loop principal de K-means:.....	3
2.1.4. Convergencia y resultado:.....	3
2.2. Explicación del algoritmo K-means concurrente.....	3
2.2.1. Inicialización:.....	4
2.2.2. Loop principal de K-means:.....	4
2.2.3. Resultado e impresión:.....	4
<b>3. Justificación del uso de los mecanismos de paralelización y sincronización utilizados.....</b>	<b>4</b>
3.1. Paralelización:.....	4
3.2. Justificación de la sincronización:.....	5
<b>4. Explicación de las pruebas realizadas y resultados.....</b>	<b>5</b>
<b>5. Explicación de la simulación realizada con promela, pegar las imágenes de evidencia.....</b>	<b>6</b>
<b>6. Explicación del análisis usando spin, pegar las imágenes de evidencia.....</b>	<b>6</b>
<b>7. Conclusiones.....</b>	<b>7</b>
<b>8. Bibliografía.....</b>	<b>7</b>

## 1. Introducción

K-means es un método de agrupación muy utilizado en la minería de datos, tiene como objetivo dividir los datos en grupos o clústeres de tal manera que los puntos en el mismo clúster sean más similares entre sí.

Nuestro objetivo es aplicar este algoritmo con el uso de goroutines en el lenguaje GO, utilizando así la programación concurrente de manera eficiente.

Utilizaremos datos aleatorios creados en el mismo código (1000000 registros) y nuestro número de clústeres será de 3 ( $K=3$ ).

## 2. Algoritmo Secuencial(K-Means) secuencial y de manera concurrente.

### 2.1. Explicación del Algoritmo Secuencial(K-Means).

El código implementa el algoritmo K-means para realizar clustering en un conjunto de puntos. Aquí un desglose de su funcionamiento:

#### 2.1.1. Generación de puntos aleatorios:

- La función `generatePoints` crea un conjunto de  $n$  puntos con coordenadas  $X$  e  $Y$  aleatorias dentro del rango  $(0, 100)$ .

#### 2.1.2. Inicialización de Clusters:

- Se define el número de clusters deseado ( $k$ ) en la función `main`.
  - La función `kmeans` inicializa  $k$  clústeres con centros aleatorios dentro del mismo rango  $(0, 100)$ .

#### 2.1.3. Loop principal de K-means:

El loop principal itera hasta que los centros de los clústeres convergen:

- **Asignación de puntos a clústeres:**
  - Para cada punto, se calcula la distancia a cada centro de clúster.
  - El punto se asigna al clúster con el centro más cercano.
- **Actualización de centros de clústeres:**
  - Se calcula la media de las coordenadas  $X$  e  $Y$  de los puntos pertenecientes a cada clúster.
  - Los centros de los clústeres se actualizan con las nuevas medias.
  - Se comprueba si los nuevos centros son iguales a los anteriores (convergencia).

#### 2.1.4. Convergencia y resultado:

- El loop termina cuando los centros de los clusters dejan de cambiar, indicando que se ha encontrado una configuración estable.
- Finalmente, se imprimen los centros de cada cluster y los puntos asignados a cada uno.

### 2.2. Explicación del algoritmo K-means concurrente.

Nuestro código proporcionado implementa el algoritmo K-means de manera concurrente para agrupar un conjunto de datos en clústeres. Aquí un desglose de su funcionamiento:

### **2.2.1. Inicialización:**

- Se genera un conjunto de datos aleatorios (data) con nSamples puntos y nFeatures características.
- Se define el número de clústeres deseados (k).
- Se inicializan los centroides (centroids) aleatoriamente a partir de los datos.
- Se crean variables para almacenar las asignaciones de puntos a clústeres (assignments) y los vectores para un procesamiento eficiente (vectors).

### **2.2.2. Loop principal de K-means:**

El algoritmo itera por un número máximo de iteraciones (maxIterations):

- **Asignación de puntos a clusters concurrente:**
  - Se utiliza un grupo de goroutines (wg) para asignar puntos a los centroides más cercanos en paralelo.
  - Cada goroutine itera sobre los puntos y calcula la distancia a cada centroide.
  - El punto se asigna al cluster con el centroide más cercano y la asignación se almacena en assignments.
- **Actualización de centroides concurrente:**
  - Se utilizan goroutines para actualizar los centroides en paralelo.
  - Se crea un mutex (mu) para evitar condiciones de carrera durante la actualización de cada centroide.
  - Se crean grupos de clusters (clusters) para almacenar los puntos asignados a cada cluster.
  - Cada goroutine calcula la media de los puntos pertenecientes a su cluster asignado y actualiza el correspondiente centroide.

### **2.2.3. Resultado e impresión:**

- Se imprimen los centroides finales.
- Se imprime un número definido de asignaciones de puntos a clusters (printAssignments).
- Se muestra el tiempo de ejecución del algoritmo.

## **3. Justificación del uso de los mecanismos de paralelización y sincronización utilizados.**

### **3.1. Paralelización:**

La asignación de puntos a clusters es una tarea independiente para cada punto, por lo que se presta naturalmente a la paralelización. Utilizar goroutines permite aprovechar múltiples núcleos de la CPU para acelerar esta fase del algoritmo, mejorando significativamente el rendimiento.

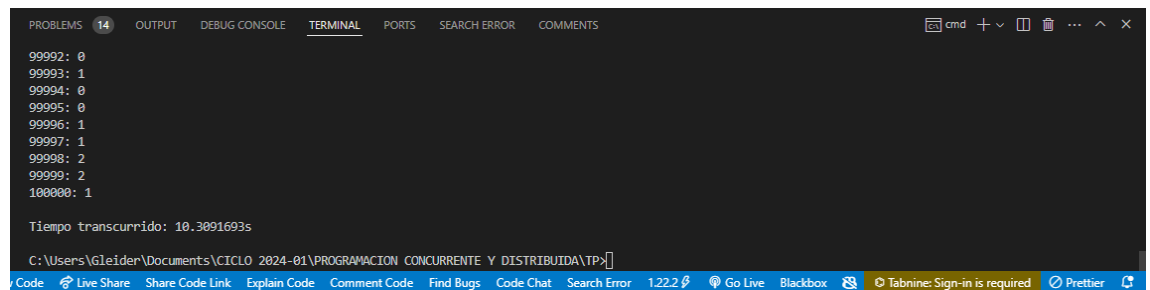
### 3.2. Justificación de la sincronización:

La actualización de cada centroide requiere sumar los puntos asignados a ese cluster y calcular la media. El mutex (mu) garantiza que solo un goroutine acceda y modifique un centroide a la vez, evitando condiciones de carrera y resultados inconsistentes. Esto asegura la integridad de los datos y la convergencia del algoritmo.

## 4. Explicación de las pruebas realizadas y resultados.

### 4.1. Algoritmo K-means en Go de manera secuencial

Este código implementa el algoritmo K-means en Go de manera secuencial. Inicializa aleatoriamente los centroides, asigna puntos de datos a los centroides más cercanos en cada iteración y actualiza los centroides con la media de los puntos asignados. Finalmente, imprime los centroides finales y las asignaciones de los puntos de datos.



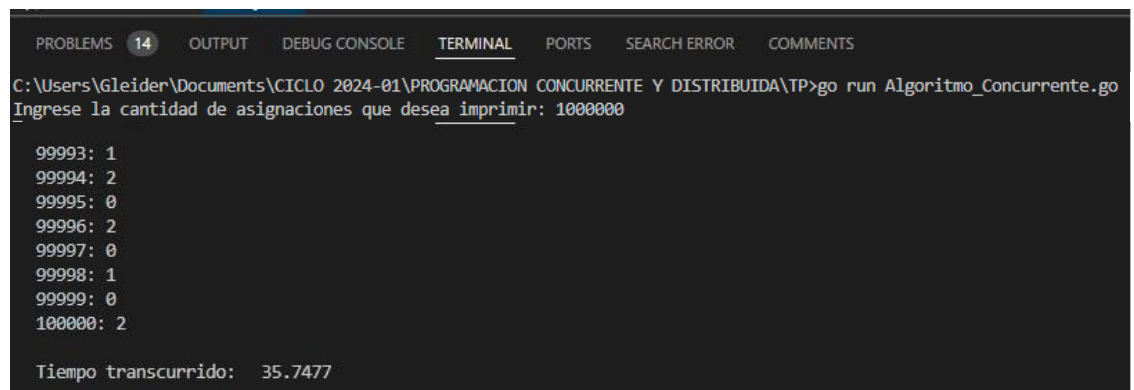
```
PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR COMMENTS
99992: 0
99993: 1
99994: 0
99995: 0
99996: 1
99997: 1
99998: 2
99999: 2
100000: 1

Tiempo transcurrido: 10.3091693s
C:\Users\Gleider\Documents\CICLO 2024-01\PROGRAMACION CONCURRENTES Y DISTRIBUIDA\TP>
```

Imagen 1 : Screenshot de la compilación del algoritmo K-Means.

### 4.2. Algoritmo K-means en Go de manera concurrente

La diferencia significativa entre esta versión concurrente y la implementación secuencial anterior radica en cómo maneja la asignación de puntos de datos y las actualizaciones de centroides. Al utilizar goroutines y primitivas de sincronización, el código realiza estas tareas de manera concurrente. Este enfoque puede mejorar potencialmente la velocidad de ejecución general aprovechando los múltiples núcleos o subprocesos disponibles en el sistema.



```
PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR COMMENTS
C:\Users\Gleider\Documents\CICLO 2024-01\PROGRAMACION CONCURRENTES Y DISTRIBUIDA\TP>go run Algoritmo_Concurrente.go
Ingrese la cantidad de asignaciones que desea imprimir: 100000

99993: 1
99994: 2
99995: 0
99996: 2
99997: 0
99998: 1
99999: 0
100000: 2

Tiempo transcurrido: 35.7477
```

Imagen 2 : Screenshot de la compilación del algoritmo K-Means Concurrente.

## 5. Explicación de la simulación realizada con promela, pegar las imágenes de evidencia.

En la simulación con promela, se definen 3 canales(data\_chanel,centroid\_chanel y convergence\_chanel) estos canales sirven para la comunicación entre los diferentes procesos. Después los procesos de “datageneratos” y “centroidinitializer” son los procesos donde se generan los datos que necesitamos y los centroides.

Después se verifican si los procesos convergieron con el proceso “convergencechecker” y envía una señal a través del canal correspondiente y por último el proceso main inicializa todos los procesos.

```
trabajoparcial.pml
1  mtype = {INIT, DATA, CENTROID, CONVERGED}
2
3  chan data_channel = [100] of {mtype, int}; // Canal para datos
4  chan centroid_channel = [10] of {mtype, int}; // Canal para centroides
5  chan convergence_channel = [1] of {mtype, int}; // Canal para señalar convergencia
6
7  active proctype DataGenerator() {
8      data_channel ! INIT, 0; // Iniciar la transmisión de datos
9      int i = 0;
10     do
11         :: data_channel ! DATA, i; // Enviar dato
12         i = i + 1;
13     od
14 }
15
16 active proctype CentroidInitializer() {
17     centroid_channel ! INIT, 0; // Iniciar la transmisión de centroides
18     int i = 0;
19     do
20         :: centroid_channel ! CENTROID, i; // Enviar centroide
21         i = i + 1;
22     od
23 }
24
25 proctype Centroid() {
26     int centroid_value;
27     bool converged = false;
28     do
29         :: centroid_channel ? CENTROID, centroid_value; // Recibir centroide
30         printf("Received centroid: %d\n", centroid_value);
31         // Lógica de actualización de centroides
32         :: convergence_channel ? CONVERGED, _; // Verificar convergencia
33         converged = true;
34     od
35 }
```

```

lupomi@DESKTOP-JICNCS3:~/concurrency$ spin trabajoparcial.pml
timeout
#processes: 8
queue 2 (data_channel): [INIT,0][INIT,0][DATA,0][DATA,0][DATA,1][DATA,2][DATA,3][DATA,1][DATA,4][DATA,2]
[DATA,5][DATA,6][DATA,3][DATA,7][DATA,4][DATA,8][DATA,5][DATA,9][DATA,6][DATA,10][DATA,7][DATA,11][DATA,8][DATA,12][DATA
,9][DATA,10][DATA,13][DATA,11][DATA,14][DATA,15][DATA,16][DATA,12][DATA,13][DATA,17][DATA,14][DATA,18][DATA,19][DATA,15]
[DATA,20][DATA,16][DATA,21][DATA,17][DATA,18][DATA,22][DATA,23][DATA,19][DATA,24][DATA,25][DATA,20][DATA,26][DATA,27][DA
TA,21][DATA,22][DATA,23][DATA,28][DATA,29][DATA,24][DATA,30][DATA,25][DATA,26][DATA,31][DATA,27][DATA,32][DATA,28][DATA,
29][DATA,33][DATA,34][DATA,30][DATA,31][DATA,35][DATA,32][DATA,36][DATA,37][DATA,33][DATA,38][DATA,39][DATA,34][DATA,40]
[DATA,35][DATA,41][DATA,36][DATA,37][DATA,38][DATA,42][DATA,39][DATA,43][DATA,40][DATA,44][DATA,41][DATA,42][DATA,45][DA
TA,43][DATA,46][DATA,44][DATA,47][DATA,45][DATA,48][DATA,46][DATA,47][DATA,49]
queue 1 (centroid_channel): [INIT,0][CENTROID,0][CENTROID,1][INIT,0][CENTROID,2][CENTROID,0][CENTROID,1]
[CENTROID,3][CENTROID,2][CENTROID,4]
queue 3 (convergence_channel): [INIT,0]
337: proc 7 (ConvergenceChecker:1) trabajoparcial.pml:48 (state 3)
337: proc 6 (Centroid:1) trabajoparcial.pml:28 (state 5)
337: proc 5 (Data:1) trabajoparcial.pml:39 (state 3)
337: proc 4 (CentroidInitializer:1) trabajoparcial.pml:19 (state 5)
337: proc 3 (DataGenerator:1) trabajoparcial.pml:10 (state 5)
337: proc 2 (Main:1) trabajoparcial.pml:60 (state 6) <valid end state>
337: proc 1 (CentroidInitializer:1) trabajoparcial.pml:19 (state 5)
337: proc 0 (DataGenerator:1) trabajoparcial.pml:10 (state 5)
8 processes created
lupomi@DESKTOP-JICNCS3:~/concurrency$

```

## 6. Explicación del análisis usando spin.

Una vez ejecutado el código con el comando “**spin -a**” se crea en nuestra carpeta varios archivos, el más importante es el archivo “**pan.c**”, luego ejecutamos el código “**gcc -o pan pan.c**” para que se pueda crear el archivo “**pan**”(Que aparece en verde en la primera imagen). luego para poder ejecutar el archivo se utiliza el comando “**./pan**”.

```

lupomi@DESKTOP-JICNCS3:~/concurrency$ spin -a trabajoparcial.pml
lupomi@DESKTOP-JICNCS3:~/concurrency$ la
-bash: la: command not found
lupomi@DESKTOP-JICNCS3:~/concurrency$ ls
break concurrency.pml nuevahoja.pml pan.b pan.c pan.h pan.m pan.p pan.t trabajoparcial.pml
lupomi@DESKTOP-JICNCS3:~/concurrency$ gcc -o pan pan.c
lupomi@DESKTOP-JICNCS3:~/concurrency$ ls
break concurrency.pml nuevahoja.pml pan pan.b pan.c pan.h pan.m pan.p pan.t trabajoparcial.pml
lupomi@DESKTOP-JICNCS3:~/concurrency$

```

Esto es lo que se obtiene cuando se ejecuta el comando “**./pan**” donde nos muestra información importante como:

- La versión del spin.
- El tamaño del vector indicado por el state-vector (988 bytes).
- La profundidad alcanzada que es lo que el spin ha recorrido hasta encontrar el Estado final (226).
- El número de errores que hay en la simulación.
- El número de estados unicos almacenados por el spin durante la simulación (227).
- El número total de transiciones exploradas en la simulación (227).
- El número de conflictos de hash resueltos (0).
- Para terminar en la parte final proporciona información estadística sobre el uso de la memoria en la simulación.

```
pan:1: invalid end state (at depth 225)
pan: wrote trabajoparcial.pml.trail

(Spin Version 6.5.2 -- 6 December 2019)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
  never claim           - (none specified)
  assertion violations   +
  acceptance cycles     - (not selected)
  invalid end states     +

State-vector 988 byte, depth reached 226, errors: 1
  227 states, stored
    0 states, matched
  227 transitions (= stored+matched)
    0 atomic steps
hash conflicts:          0 (resolved)

Stats on memory usage (in Megabytes):
  0.220    equivalent memory usage for states (stored*(State-vector + overhead))
  0.484    actual memory usage for states
128.000    memory used for hash table (-w24)
  0.534    memory used for DFS stack (-m10000)
128.925    total actual memory usage
```



## 7. Conclusiones.

- El uso de goroutines nos permite asignar puntos a los centroides de manera concurrente, acelerando así el proceso de asignación en comparación con un enfoque secuencial.
- Al utilizar concurrencia y goroutines se puede reducir significativamente el tiempo de ejecución del algoritmo, incluso cuando se trabaja con datos grandes como es el caso.
- Se concluye que el algoritmo K-means concurrente ofrece un mayor potencial de rendimiento y escalabilidad en sistemas con múltiples núcleos. Sin embargo, es importante considerar las necesidades específicas de la aplicación, la complejidad de la implementación y los recursos disponibles al elegir entre el algoritmo secuencial y el concurrente.

## 8. Link del repositorio de Github y del video.

A continuación se muestra el link del repositorio:  
<https://github.com/GleiderCastro/Aplicacion-en-GO---PCYD.git>.

Link del video: <https://youtu.be/O0qQkNbNE7k>

## 9. Bibliografía

- Likebupt. (2023, 1 junio). *Agrupación en clústeres K-means: referencia del componente* - Azure Machine Learning. Microsoft Learn.  
<https://learn.microsoft.com/es-es/azure/machine-learning/component-reference/k-means-clustering?view=azureml-api-2>
- Okereke, G. E., Bali, M. C., Okwueze, C. N., Ukekwe, E. C., Echezona, S. C., & Ugwu, C. I. (2023). *K-means clustering of electricity consumers using time-domain features from smart meter data. Journal of Electrical Systems and Information Technology*, 10(1), 2.  
doi:<https://doi.org/10.1186/s43067-023-00068-3>