

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**FACULTAD DE INGENIERÍA**

**CARRERA PROFESIONAL**

CIENCIAS DE LA COMPUTACIÓN

**CURSO**

Programación Concurrente y Distribuida (CC65)

Práctica Calificada 2 2025-1

**ESTUDIANTE**

Campos Espinoza, Jak Cristian (u20211a184)

**DOCENTE**

Jara García, Carlos Alberto

**Lima, 02 de mayo del 2025**

1. **INTRODUCCIÓN**

La programación concurrente permite realizar múltiples tareas en paralelo, lo cual es esencial en aplicaciones modernas que manejan grandes volúmenes de datos, como en el aprendizaje automático. En esta práctica, se desarrolló el algoritmo de regresión lineal de forma secuencial y concurrente en el lenguaje Go, aprovechando las goroutines para mejorar el rendimiento en conjuntos de datos extensos. Se busca comparar ambos enfoques en términos de precisión, rendimiento y eficiencia.

1. **MARCO TEÓRICO**

La programación concurrente es un paradigma de desarrollo que permite ejecutar múltiples procesos o tareas al mismo tiempo, con el fin de mejorar el rendimiento, escalabilidad y aprovechamiento de recursos en sistemas computacionales. Este enfoque es especialmente útil en la era del cómputo intensivo, donde se requiere procesar grandes volúmenes de datos de forma eficiente (Silberschatz, Galvin & Gagne, 2020).

En el lenguaje de programación Go, la concurrencia se implementa mediante goroutines, que son funciones ligeras que se ejecutan de manera concurrente con otras funciones dentro del mismo espacio de memoria. Estas permiten aprovechar múltiples núcleos del procesador sin incurrir en la complejidad típica de los hilos del sistema operativo (Donovan & Kernighan, 2016). Para la coordinación de goroutines, Go proporciona primitivas como los canales (chan) y los grupos de espera (sync.WaitGroup), que facilitan la sincronización y el paso seguro de datos.

En el contexto del aprendizaje automático, la regresión lineal es un algoritmo fundamental que permite modelar la relación entre una variable independiente y una dependiente , asumiendo una relación lineal entre ambas. La fórmula general del modelo es:

donde representa la pendiente y la intersección con el eje . Este algoritmo se utiliza ampliamente por su simplicidad y eficiencia, y es particularmente efectivo en tareas de predicción y análisis de tendencias (James et al., 2021).

Cuando se trabaja con algoritmos concurrentes, es crucial identificar secciones críticas, es decir, zonas del código donde múltiples goroutines acceden a recursos compartidos. Si no se maneja correctamente este acceso concurrente, se pueden producir condiciones de carrera, donde el resultado del programa depende del orden de ejecución de las goroutines. En Go, estas condiciones se mitigan usando estructuras como sync.Mutex o evitando completamente el acceso compartido mediante el uso de canales (Go Authors, 2024).

El uso de programación concurrente en regresión lineal permite dividir el conjunto de datos en bloques, procesarlos en paralelo y luego combinar los resultados parciales para obtener los coeficientes finales del modelo. Esto reduce significativamente el tiempo de ejecución sin afectar la precisión del modelo, siempre que se controle adecuadamente el acceso a las variables compartidas.

1. **METODOLOGÍA**

**Prompts usados**

"¿Cómo implementar regresión lineal concurrente en Go sin bibliotecas externas?"

"¿Cómo usar goroutines y WaitGroup para dividir tareas en Go?"

"¿Cómo detectar condiciones de carrera en código concurrente en Go?"

**Herramientas utilizadas**

* ChatGPT para comprensión del enfoque concurrente
* Documentación oficial de Go: <https://go.dev/>
* Editor Visual Studio Code
* Consola de comandos para medir tiempos de ejecución

1. **DESARROLLO**

**Versión Secuencial**

El código regresionLSec.go realiza el cálculo de los coeficientes de regresión sumando todos los valores en un solo bucle.

**Versión Concurrente**

El código regresionLCon.go divide los datos entre múltiples goroutines, cada una de las cuales calcula sumas parciales que luego son agregadas usando canales y un WaitGroup.

1. **RESULTADOS**

**Secuencial**

Tiempo: 6.0127ms

Pendiente: 1.999856

Intersección: 10.009678

**Concurrente**

Tiempo: 2.8482ms (con 8 goroutines)

Pendiente: 1.999935

Intersección: 10.003010

**Comparación de Rendimiento**

|  |  |  |
| --- | --- | --- |
| **Versión** | **Tiempo (ms)** | **Diferencia (%)** |
| Secuencial | 6.0127 | - |
| Concurrente | 2.8482 | ~52.63% más rápida |

**Problemas detectados y soluciones**

Se identificó la necesidad de evitar condiciones de carrera al acumular resultados, lo cual fue resuelto enviando estructuras **Parcial** por canal y evitando variables compartidas sin sincronización.

1. **ANÁLISIS Y DISCUSIÓN**

Eficiencia: La versión concurrente redujo significativamente el tiempo de ejecución con un dataset de 2 millónes de datos.

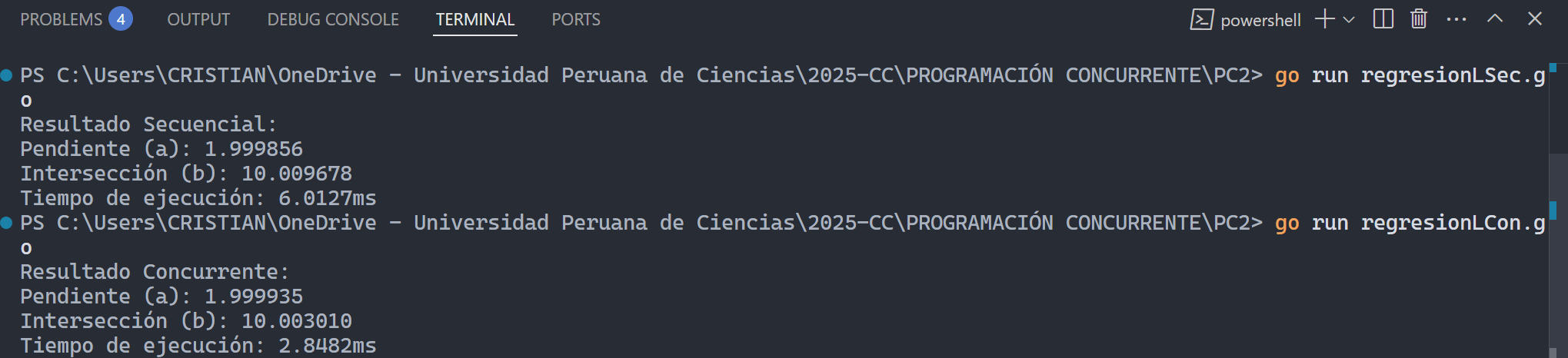
Escalabilidad: El rendimiento mejora con mayor número de goroutines, aunque puede llegar a un punto de saturación si se usan demasiadas.

Legibilidad y complejidad: La versión concurrente es más compleja de mantener, pero ofrece beneficios en aplicaciones de procesamiento masivo.

1. **CONCLUSIONES**

* El uso de goroutines en Go para procesamiento paralelo ofrece mejoras significativas en rendimiento.
* El manejo adecuado de secciones críticas evita errores y asegura precisión en los resultados.
* Esta práctica demuestra la aplicabilidad real de la programación concurrente en tareas de machine learning con grandes volúmenes de datos.
* El enfoque concurrente es especialmente útil en sistemas distribuidos o de alto rendimiento.

1. **EVIDENCIA**



1. **ENLACE GITHUB**

<https://github.com/JakCE/Progra-Concurrente/tree/main/PC2>

1. **REFERENCIAS BIBLIOGRÁFICAS**

Donovan, A. A., & Kernighan, B. W. (2016). The Go programming language. Addison-Wesley Professional. <https://www.gopl.io/>

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts (10th ed.). Wiley. <https://os.ecci.ucr.ac.cr/slides/Abraham-Silberschatz-Operating-System-Concepts-10th-2018.pdf>

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An introduction to statistical learning: with applications in R (2nd ed.). Springer. <https://www.statlearning.com/>

The Go Authors. (2024). The Go programming language specification. Go.dev. <https://go.dev/ref/spec>