Algoritmos de Inteligencia Artificial y Machine Learning en Go

José M. Miranda Villagrán

Instituto Tecnológigo de Veracruz
mirandav@itver.edu.mx

Salvador E. Venegas-Andraca

Tecnológico de Monterrey, Escuela de Ingeniería y Ciencias
salvador.venegas-andraca@keble.oxon.org

Gabriel Acosta

Instituto Politécnico Nacional
algo@algo.com

August 19, 2019

Abstract

En este articulo se explica la etapa actual en la que se encuentra desarrollada la computación y por qué necesitamos un nuevo lenguaje como Go para aprovechar los avances actuales del hardware.

I. Introducción

En los últimos años, ha surgido un nuevo lenguaje de programación: Go o GoLang. Se presentó el 10 de noviembre de 2009 como un nuevo lenguaje de programación con tiempos de compilación muy rápido. Las excelentes herramientas de Go, el elegante modelo de concurrencia y el enfoque único de la orientación a objetos ha cautivado la atención de los desarrolladores tanto de los lenguajes compilados como de los scripts.

i. Limitaciones de hardware

El primer procesador Pentium 4 con velocidad de reloj de 3.0 GHz[1] fue presentado en 2004 por Intel. Actualmente los equipos comerciales tienen una velocidad de reloj de 2.30GHz. Entonces, casi en una década, no hay demasiada ganancia en el poder de procesamiento en bruto. Se puede ver la comparación de aumento de potencia de procesamiento con respecto al tiempo en la figura 1.

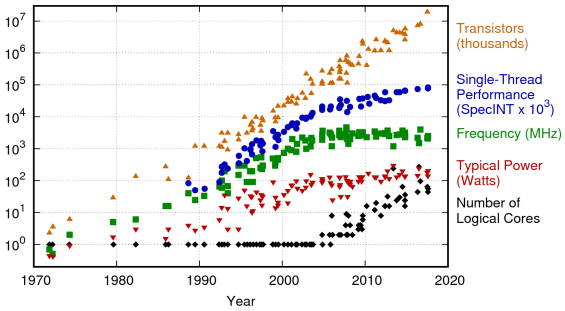

Figure 1: *42 años de tendencias de microprocesadores.*

En el gráfico anterior puede ver que el rendimiento de un solo hilo y la frecuencia del procesador se mantuvieron estables durante casi una década. Agregar más transistores no es la solución. Esto se debe a que

[1] `http://www.informit.com/articles/article.aspx?p=339073`

a menor escala comienzan a surgir algunas propiedades cuánticas y porque en realidad cuesta más poner más transistores.

Entonces, para la solución del problema anterior los fabricantes comenzaron a agregar más núcleos al procesador, también introdujeron hyper-threading y se agregó más caché al procesador para aumentar el rendimiento.

Pero las soluciones anteriores también tienen sus propias limitaciones. No podemos agregar más caché al procesador para aumentar el rendimiento ya que el caché tiene límites físicos: cuanto más grande es el caché, más lento se vuelve[2]. Agregar más núcleos al procesador también tiene su costo. Estos procesadores multinúcleo pueden ejecutar varios procesos simultáneamente y eso brinda concurrencia.

Por lo tanto, si no podemos confiar en las mejoras de hardware, el único camino a seguir es un software más eficiente para aumentar el rendimiento. Pero lamentablemente, los lenguajes de programación modernos no son muy eficientes.

ii. ¿Por qué Go?

> "Go will be the server language of the future." - Tobias Lütke[3], Shopify

Para evaluar el lenguaje Go como una solución para el desarrollo web moderno, hay que ver las tendencias y considerar que debería proporcionar "el lenguaje servidor del futuro".[4]

Los fabricantes de hardware están agregando cada vez más núcleos a los procesadores para aumentar el rendimiento. Todos los centros de datos funcionan con esos procesadores. Además, las aplicaciones actuales utilizan múltiples microservicios para mantener conexiones de bases de datos, colas de mensajes y mantener cachés. Por lo tanto, el software que se desarrolla y los lenguajes de programación deben admitir la concurrencia fácilmente y deben ser escalables con un mayor número de núcleos.

Pero, la mayoría de los lenguajes de programación modernos (como Java, Python, etc.) son del entorno single-threaded de los años 90. La mayoría de esos lenguajes de programación admite multi-threaded. Pero el verdadero problema viene con la ejecución concurrente, threading-locking, las condiciones de carrera[5] y los deadlock. Esas cosas hacen que sea difícil crear una aplicación multi-threaded en esos lenguajes.

Por ejemplo, crear un nuevo hilo en Java no es eficiente en la memoria. La sobrecarga real varía según las plataformas, pero la creación de hilos lleva tiempo, ya que introduce latencia en el procesamiento de solicitudes y requiere cierta actividad de procesamiento por parte de JVM y SO. Si las solicitudes son frecuentes y ligeras, como en la mayoría de las aplicaciones de servidor, la creación de un nuevo hilo para cada solicitud puede consumir importantes recursos informáticos. Además, si desea comunicarse entre dos o más hilos, es muy difícil [Peierls, 2006, pág 73].

II. Algoritmos

i. Algoritmos de búsqueda en grafos

En general existen dos categorías para clasificar los distintos algoritmos para búsqueda en grafos:

- Búsqueda sin información.
- Búsqueda informada.

La **búsqueda sin información del dominio**, también llamada ciega o exhaustiva es:

- **Sistemática:** No deja sin explorar ningún nodo y lo explora sólo una vez.
- **Objetiva:** Pues no depende del dominio del problema.

Estas técnicas son poco eficientes, aunque, en coste promedio, son mejores que las que utilizan el conocimiento del dominio del problema. Se caracterizan, entre otras cosas, porque la aplicación de los operadores a los estados

[2] https://www.researchgate.net/post/Why_is_the_capacity_of_of_cache_memory_so_limited

[3] https://en.wikipedia.org/wiki/Tobias_Lütke

[4] https://nathany.com/why-go/

[5] https://searchstorage.techtarget.com/definition/race-condition

se realiza de manera sistemática, uno detrás de otro (estrategia de control no informada). Además todas ellas son búsquedas por tentativas y utilizan un esquema de producción (búsqueda en el espacio de estados).

La estrategia de **búsqueda informada** (la que utiliza el conocimiento específico del problema más allá de la definición del problema en sí mismo)puede encontrar soluciones de una manera más eficiente que una estrategia no informada.
Como criterio para diferenciarlas, se utilizará el orden en que se recorren los estados. Así, para el espacio de estados, tendremos:

- Búsqueda en amplitud.
- Búsqueda en profundidad.
- Búsqueda A*.

i.1 Búsqueda en amplitud (BFS)

La búsqueda en amplitud es una estrategia sencilla en la que se expande primero el nodo raíz, a continuación se expanden todos los sucesores del nodo raíz, después sus sucesores, etc. En general, se expanden todos los nodos a una profundidad en el árbol de búsqueda antes de expandir cualquier nodo del próximo nivel. La forma de implementarlo es poner los sucesores de cada nodo al final de una cola. El pseudocódigo se da en la figura **??**.

References

[Peierls, 2006] Peierls, T., Goetz, B., Bloch, J., Bowbeer, J., Lea, D. and Holmes, D. (2006). *Task Execution*. En Java Concurrency in Practice(424) Universidad de Michigan: Pearson Education.