Hash: 1677

Treball Dirigit: Concurrència en Go

1. Introducció

En aquest document discutirem en una primera instància que és la concurrència i com es pot gestionar des del llenguatge de programació Go o Golang. Posarem en context que és Go i explicarem quins usos i avantatges té sobre altres llenguatges. En una segona part revisarem la bibliografia i la qualitat d'aquesta.

Aquest mateix document es complementa amb una presentació gravada pel mateix estudiant. A més, compleix els objectius de Comunicació eficaç oral i escrita G4.3 i Ús solvent dels recursos d'informació G6.3.

2. Concurrència

2.1 Definició

Per començar definirem que és la concurrència des de diferents fonts. Segons diccionari.cat la concurrència és un aplec format per les persones que coincideixen voluntàriament en un lloc o la coincidència en el temps. Segons la Reial Acadèmia Espanyola és l'acció i efecte de concórrer; el conjunt de persones que assisteixen a un acte o reunió; la coincidència, concurs simultani de diverses circumstàncies o l'assistència, participació. Per acabar, segons Wikipedia, la concurrència en l'àmbit de la informàtica es refereix a la capacitat de diferents parts d'un programa, algoritme, o problema de ser resolt en desordre o en ordre parcial, sense afectar el resultat final.

La definició que més s'adapta seria la de coincidència en el temps, diverses coses que passen alhora. Igualment els aplecs de persones o de circumstàncies les podem entendre com fets independents (programes) que ocorren en un mateix instant. Per tant, la idea és semblant, en informàtica fem referència a poder executar diverses coses alhora. Cal tenir en compte que no ha de ser estrictament simultani, però han d'estar passant en el mateix moment o fer-ho veure.

2.2 Implicacions en Informàtica

En l'àmbit de la informàtica es pot descriure les implicacions de la concurrència amb el famós problema del sopar de filòsofs exposat per Edsger Dijkstra. Aquest problema descriu com uns filòsofs tenen un plat de fideus i una forquilla a la seva dreta. Hi ha una forquilla per cada filòsof, però cada filòsof necessita dos per menjar, ja que són fideus. Això vol dir que no poden menjar tots alhora, no hi ha prou

forquilles. Cada un pot estar una estona sense menjar, però si no mengen en molta estona es mor. Sense entrar en detalls podem abstraure la idea que hi ha una feina a realitzar però uns recursos o eines limitats.

En el món de les ciències de la computació podem veure un parell d'exemples clars per veure les implicacions que té. Un primer exemple és la divisió de problemes grans o complexos, la idea és dividir els problemes, algorismes o programes en parts més petites i fàcils de mantenir. Aquestes parts, si es pot, s'executarien alhora i tindríem programes concurrents. La concurrència es podria aconseguir en una mateixa màquina o en diverses, com passa amb els sistemes distribuïts. Un altre cas flagrant són els sistemes operatius de propòsit general on amb facilitat hi han més programes en execució que recursos disponibles. En aquests casos s'apliquen tècniques de planificació per mitigar els problemes.

2.3 Concurrència vs Paral·lelisme

En el camp de la informàtica és habitual confondre la concurrència amb el paral·lelisme. Sent el paral·lelisme una forma de concurrència no són exactament el mateix. En la concurrència fem tasques que es poden sobreposar en el temps, en canvi en el paral·lelisme les executem en el mateix instant. El paral·lelisme sobretot fa referència a executar i coordinar diferents nuclis d'un processador perquè facin una tasca conjunta. Això seguiria sent concurrent, però la concurrència també engloba processos que no són depenent entre ells ni que s'hagin de dividir en execucions simultànies. Un sistema operatiu pot tenir només un nucli i estar executant tasques concurrents, és a dir, hi ha diversos processos que requereixen "menjar" tot i no només tenir un processador disponible.

3. Go

El llenguatge de programació conegut com a Go o Golang és un projecte desenvolupat per Google, concretament per Robert Griesemer, Rob Pike i Ken Thompson. Va començar a desenvolupar-se el 2007 i va sortir a la llum el 2009, es segueix actualitzant fins a l'actualitat. El seu objectiu és mantenir una sintaxi com la de la família C tot aprofitant els avantatges de llenguatges interpretats com Python. Go és un llenguatge imperatiu i compilat, com a característiques principals destaca l'èmfasi en la concurrència i la gestió de memòria. Per ara suporta les arquitectures i386, amd64 i ARM. També cal destacar que és un llenguatge de codi obert amb llicència BSD.

El sistema de tipat és fort i estàtic, tot i que hi ha inferència de tipus. Això vol dir que el compilador infereix un tipus a una variable i ja no pot canviar dinàmicament de tipus. Aquest llenguatge permet compilar-se entre plataformes de forma nativa. A més, per fer la gestió de memòria i aïllar-la de l'usuari utilitza un recollidor de memòria brossa juntament amb l'eliminació dels punters i la seva aritmètica. Cal destacar que Go no implementa excepcions i la majoria de

convencions de la programació orientada a objectes. Per acabar cal destacar que està orientat a treure el màxim rendiment de màquines amb diversos nuclis.

4. Concurrència en Go

Com ja hem comentat un dels punts forts de Go és la concurrència. En Golang podem indicar quines instruccions o funcions seran concurrents i es podran executar en paral·lel. A més, tot això es reforça amb un sistema d'esdeveniments que permet la comunicació asíncrona entre aquestes rutines. Aquesta funcionalitat permet assolir sistemes basats en esdeveniments. El llenguatge inclou en la mateixa sintaxi eines i estructures per crear rutines o *goroutines*, com s'anomenen en el llenguatge.

4.1 Goroutines

La sintaxi principal per crear execucions simultànies és la goroutine. Qualsevol funció es pot cridar amb el prefix *go* (*go laMevaFuncio()*) per indicar que la seva execució serà concurrent, és a dir, serà una nova rutina. El llenguatge no especifica com s'han d'implementar ni com s'executaran a no ser que ho especifiquem amb mètodes de sincronització. Les rutines es traduiran a programes del sistema operatiu on s'està executant i es gestionaran a nivell de thread.

4.2 Canals

Els canals són elements clau en la concurrència de Go, permeten comunicar de forma asíncrona diferents rutines. El funcionament és molt simple, un canal és un tipus simple de Go escrit com *chan* on hem d'especificar el tipus de dades del canal. Cada canal pot transferir informació d'un únic tipus, ja que és un llenguatge fortament tipat. Per exemple un canal de paraules es declara com *chan string*. Qualsevol rutina que tingui en el seu àmbit de visibilitat aquesta variable podrà escriure i llegir d'aquest canal. Per llegir un canal només cal afegir els caràcters <- davant de la variable, per exemple <- *canal*. Per escriure en el canal s'escriu primer el canal, després els caràcter especials i al final el contingut (*canal* <- *“contingut”*).

La transmissió d'aquests missatges farà que les coronacions siguin bloquejants, cada canal funciona com una cua FIFO. Si no hi ha contingut i alguna rutina està esperant un valor quedarà bloquejada fins que rebi un primer valor. De la mateixa manera si alguna dada residual ha quedat al buffer la llegirà el següent programa que llegeixi del canal. Per això cal programar correctament els canals i les subrutines aïllant adequadament.

4.3 Planificador de Go

En general, quan parlem de concurrència en un llenguatge de programació de propòsit general pensem en processos a nivell de sistema operatiu. És a dir, llibreries que permeten crear des d'un mateix programa diferents processos al

Sistema operatiu que s'està executant. En Go la idea és semblant però amb una capa d'abstracció mes per poder mantenir un control exhaustiu sobre les rutines.

En Go s'utilitza un aparellament *M-a-N* que permet reduir el sobrecost de la creació de múltiples programes en un mateix sistema. Aquest aparellament crea diversos processos amb múltiples threads controlats per un programa conegut com *runtime*. Aquest runtime decideix en temps d'execució quants processos són necessaris i quan threads necessitarà per executar les rutines programades. En resumits comptes el *runtime* permet definir la granularitat de la concurrència i evitar així sobrecostos de canvis de context.

Llavors, les rutines es distribueixen entre una cua de gorrutinas local específica per a cada un dels processadors i una cua de rutines global. L'algoritme que defineix el comportament de cada procés és el següent, primer un cop cada seixanta-un mirarà d'agafar una rutina de la cua global, després executarà una rutina de la cua local (si hi ha) i si no intentarà agafar una altra rutina d'un altre procés de Go. En cas de no trobar cap rutina encara tornarà a mirar la cua global, si no hi ha cap finalment preguntarà a la xarxa. De manera que cada vegada que es troba una rutina que es pugui executar, s'executarà fins que quedi bloquejada de nou.

4.4 Adequació per a la programació paral·lela

Tot i que les funcions de simultaneïtat de Go no estan dirigides principalment al processament en paral·lel, es poden utilitzar per programar màquines multiprocessador de memòria compartida. Un estudi va comparar la mida del codi i la velocitat dels programes escrits per un programador experimentat que no coneixia el llenguatge i les correccions d'aquests programes per part d'un expert de Go. L'estudi va trobar que els no experts tendien a escriure algoritmes de dividir i vèncer amb recursivitat, mentre que l'expert en Go escrivia programes de sincronització amb rutines per processador. Els programes dels experts solien ser més ràpids, però també més llargs.

4.5 La manca de seguretat en situacions de competició

No hi ha restriccions sobre com les goroutines accedeixen a les dades compartides, cosa que fa possible les situacions de competició o data races. En concret, tret que un programa es sincronitzi explícitament mitjançant canals o altres mitjans, les escriptures des d'una *goroutina* no tenen garanties sobre l'ordenació en la qual seran llegides. A més, les estructures de dades internes de Go, com ara les taules de hash, no són immunes a les situacions de competició, de manera que es pot vulnerar la seguretat del tipus i de la memòria en programes multithread que modifiquen instàncies compartides d'aquests tipus sense sincronització. En lloc de suport lingüístic, com apunten els experts, la programació simultània segura es basa en convencions.

5. Aplicacions

A partir de les eines exposades es poden construir sistemes simultanis com grups de treballadors o *workers*, segmentació, on es descomprimeix i analitza un fitxer a mesura que es descarrega, crides en segon pla amb temps d'espera i altres. Els canals també tenen utilitat més enllà de la comunicació entre processos, per exemple es fan servir com a llistes segures de memòria intermèdia reciclada segura, coroutines o implementar iteradors.

Per posar casos reals de projectes i sistemes que utilitzen Go i concretament la seva concurrència parlarem de Dropbox i SendGrid. Actualment les grans empreses de tecnologia tenen problemes d'escalabilitat i utilitzen sistemes distribuïts amb diferents llenguatges i tecnologies simultanis. Go presenta l'avantatge d'esprémer al màxim els recursos disponibles de forma eficient amb grans càrregues de treball. Per exemple, Dropbox va mirar el nucli del seu emmagatzematge al núvol de Python a Go. SendGrid una companyia de missatgeria de correu electrònic ha apostat per Go a l'hora de gestionar els seus més de 500 milions de correus diaris.

6. Us i Exemples

En aquest apartat exposarem dos exemples senzills de concurrència en Go per entendre les *goroutines* i els canals.

6.1 Rutines

```go
package main

import (
    "fmt"
    "time"
)

func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

func main() {

    f("direct")

    go f("goroutine")
```

```
    go func(msg string) {
        fmt.Println(msg)
    }("going")

    time.Sleep(time.Second)
    fmt.Println("done")
}
```

En veure aquest codi per primer cop ens sembla molt similar a C en la sintaxi però no en el contingut. Podem veure com hi ha un mètode *main()* que indica el punt d'entrada del programa i una funció *f* que imprimeix per pantalla tres cops el paràmetre i el número de la iteració. En el main podem veure com primer crida la funció de forma seqüencial, després la cridem amb una *goroutine* i finalment cridem una funció anònima que imprimeix el paràmetre. Al final espera un segon i indica que ha acabat.

Si executéssim el programa diverses vegades veuríem que no sempre dóna el resultat en l'ordre programat. Això és una conseqüència d'utilitzar concurrència. També és una percepció que ens permet afirmar que s'està executant de forma concurrent.

6.2 Canals

```
package main

import (
    "fmt"
    "time"
)

func readword(ch chan string) {
    fmt.Println("Type a word, then hit Enter.")
    var word string
    fmt.Scanf("%s", &word)
    ch <- word
}

func timeout(t chan bool) {
    time.Sleep(5 * time.Second)
    t <- false
}

func main() {
    t := make(chan bool)
    go timeout(t)

    ch := make(chan string)
```

```
    go readword(ch)

    select {
    case word := <-ch:
        fmt.Println("Received", word)
    case <-t:
        fmt.Println("Timeout.")
    }
}
```

Aquest codi aplica el patró de timeout on espera una entrada de dades en menys de cinc segons o avorta. Per cada subrutina concurrent crea un canal i espera una resposta. Depenent de qui respongui primer tindrà un comportament diferent en l'estructura *select - case*. La funció *timeout* respondrà fals al cap de cinc segons i la funció *readword* respondrà tan d'hora com un usuari introdueixi una paraula.

7. Avantatges i Inconvenients

Avantatges	Inconvenients
• És un llenguatge ràpid en termes d'execució i compilació. • Al ser semblant a C és fàcil d'aprendre i entendre. • Les interfícies són realment útils per desenvolupar i testejar. • La llibreria estàndard és rica i completa. • Gestió de memòria simplificada. • Concurrència simplificada.	• No existeixen els tipus genèrics. • Estructures típiques de llenguatges orientats a objectes. • Moltes llibreries no tenen suport en Go. • La gestió de dependències no és prou consistent.

8. Comparació amb altres LPs

En principi qualsevol llenguatge de programació pot ser concurrent i implementar threads o sistemes similars. Els llenguatges més populars com C++ o Java utilitzen llibreries estàndard mentre que Go és implícit en la sintaxi. A priori l'avantatge de Go és el seu *runtime* que optimitza l'execució. Això no vol dir que els altres llenguatges no puguin tenir llibreries similars i arribar a rendiments iguals. De fet la diferència és aquesta, mentre en uns llenguatges has de buscar i aprendre de

llibreries i conceptes de sistemes operatius Go ho té implementat de forma nativa i sense preocupar-se de si el programador sap tota l'arquitectura que hi ha sota.

9. Conclusió i Visió Personal

Com a conclusió podem afirmar sense cap reticència que Go és un llenguatge de programació pensat i implementar per ser senzill i concurrent. El seu objectiu és esprémer al màxim els recursos de les màquines en programes costosos. La concurrència és fàcil d'entendre i utilitzar pel que fa a codi, a més de tota la infraestructura ja inclosa.

Pel que fa a la meva opinió com a programador no m'agraden els llenguatges amb tipatge fort, tot i que no em fa tirar enrere. El que em semblaria difícil d'ignorar és la falta d'estructures típiques dels llenguatges orientats a objectes com herència o polimorfisme. M'agradaria utilitzar el llenguatge en projectes petits com una API i provar la seva concurrència. També vull destacar que mai m'he enfrontat a problemes que requereixin tanta computació que m'hagi de plantejar quina tecnologia utilitzar.

10. Bibliografia

1. https://ca.wikipedia.org/wiki/El_sopar_de_fil%C3%B2sofs
2. https://en.wikipedia.org/wiki/Concurrency_(computer_science)
3. https://en.wikipedia.org/wiki/Concurrent_computing
4. https://dle.rae.es/concurrencia
5. http://www.diccionari.cat/lexicx.jsp?GECART=0033308
6. https://blog.friendsofgo.tech/posts/concurrencia-en-golang/
7. https://en.wikipedia.org/wiki/Go_(programming_language)
8. http://lsi.vc.ehu.es/pablogn/docencia/manuales/SO/TemasSOuJaen/INTRODUCCION/6Clasificaciondelossistemasoperativos.htm
9. https://www.oscarblancarteblog.com/2017/03/29/concurrencia-vs-paralelismo/
10. https://www.youtube.com/watch?v=mXpZDQ2Au8U
11. https://tour.golang.org/concurrency/1
12. https://gobyexample.com/channels
13. https://blog.friendsofgo.tech/posts/go-scheduler/
14. https://morsmachine.dk/go-scheduler

15. https://www.softkraft.co/companies-using-golang/
16. https://willowtreeapps.com/ideas/the-pros-and-cons-of-programming-in-go
17. https://www.cpp-vs.com/versus/go-process-coordination
18. https://golangdocs.com/concurrency-in-golang
19. https://go.dev/
20. https://golang.org/
21. http://www.findingdulcinea.com/news/education/2010/march/The-Top-10-Reasons-Students-Cannot-Cite-or-Rely-on-Wikipedia.html
22. https://www.youtube.com/watch?v=XzU73kRq_kk

11. Descripció de les fonts d'informació emprades

En aquest treball només he utilitzat fonts d'internet donat que no disposo de llibres o referències de Go en llibre a casa. La possibilitat d'anar a una biblioteca la he valorat, però no és comparable amb la comoditat dels cercadors. A més, no disposo de temps per llegir llargs manuals. Per la tipologia del treball era més ràpid i eficient anar a buscar fonts d'informació que parlessin directament del tema que necessitava.

Durant el meu recorregut com a programador i desenvolupador m'he adonat que les fonts més útils són aquelles que parlen de temes concrets i no fan llargues dissertacions sobre temes monstruosos. Quan busco un tema o problema vull una solució a aquest, no una explicació de com funciona tot un sistema. Si vull investigar per pura curiositat o aprenentatge si val la pena dedicar un temps i deixar-se portar. Per tant, pàgines com Quora o StackOverflow són una mina de píndoles d'informació ràpides i nutritives. Tots ja sabem que no som els primers que tenim aquell problema i per tant només cal fer una cerca al nostre indexador preferit.

Les fonts en el treball exposat son bàsicament pàgines de Wikipedia on trobes la informació estructurada i verificada per una gran comunitat. A més, he consultat blogs i algun vídeo de Youtube que donen informació d'algun tema en concret com el *runtime* de Go o els canals en el mencionat llenguatge. Com no podia ser d'altra forma la informació s'ha contrastat en diverses fonts, ja que jo no sóc coneixedor de primera mà dels temes tractats. Sorprenentment la documentació oficial del llenguatge no m'ha ajudat gaire a entendre certes parts i he optat per utilitzar fonts de tercers.

12. Avaluació de la qualitat de la informació trobada

En aquest apartat partiré de la premissa que en buscar informació sobre alguna eina no li interessa a ningú fer perdre el temps amb informació falsa. A més que és fàcilment comprovable en el nostre cas mirant si Go funciona com realment indiquen les nostres fonts. Wikipedia no és considerat un bon lloc per referenciar la informació, però si és un bon lloc de partida per posar-se en context i començar a indagar en el tema. Per tant considerarem les referències a Wikipedia de baixa qualitat. Per altra banda els blogs especialitzats a difondre i ensenyar sobre Go són d'alta qualitat, ja que és contraproduent explicar coses falses i són fàcilment verificables. També podem dir que utilitzar pàgines que tinguin referències en elles ajuden a donar veracitat si sabem d'on treu la informació i que l'ha donat.

Per acabar, tenint en compte la naturalesa del nostre tema, la millor font d'informació és la documentació oficial del llenguatge. Encara que existeixi documentació de tercers sabem que la documentació és una eina imprescindible per a l'ús del llenguatge, per fer una analogia és el manual d'instruccions de la nostra joguina. A vegades pot passar que s'entengui millor la d'altres fonts, però sempre ens hem de recolzar en la que presenta el creador de la tecnologia.