Go (Golang) - The missing language for the 21st century

Codemotion 2015
17 December 2015

Uri Shamay
Lead Developer, Juno

The Go programming language

- Written by Google
- Written in the 21st century :: Design: 2007, Open Source: 2009, Stable Go 1.0: 2012
- 6 Years old
- Influenced by: C, occam, Limbo, Modula, Newsqueak, Oberon, Pascal, Smalltalk
- Current Version: Go 1.5.1

Continue..

- Static types but looks like a script
- Compiled, concurrent, imperative, structured
- Functions can return multiple value
- No classes, has structs
- No exceptions, has errors
- OOP based on interfaces & composition
- Visibility is at the package level. No public/protected/package/private.
- Has pointers to control memory layout for high performance, **BUT - no buffer overflows, no pointer arithmetic**
- Memory efficient: concrete primitives: int & uint **BUT also** int8 & uint8, float32 & float64
- Garbage-collected: since Go 1.5 - GC latencies well below the 10 millisecond

Continue...

- Rich standard library
- Compiles to native code, statically linked, and since Go 1.5 also dynamically
- Blazing fast compilation - the whole SDK took tens of seconds
- Easy to deploy one static binary without any dependencies
- Cross platform compilation
- ABI (Application Binary Interface) - compile on one Linux distro, and run in any Linux distro
- General purpose: Tools & Utilities, Cloud Infrstructure, IOT (Raspberry Pi & more), Mobile, Web
- Google App Engine support Python & Java, now also Go applications

Continue....

Supported OS ($GOOS) & ARCH ($GOARCH)

- darwin (386, amd64, arm, arm64)
- dragonfly (amd64)
- freebsd (386, amd64, arm)
- linux (386, amd64, arm, arm64, ppc64, ppc64le)
- netbsd (386, amd64, arm)
- openbsd (386, amd64, arm)
- plan9 (386, amd64)
- solaris (amd64)
- windows (386, amd64)

Motivation for Go

Started as an answer to software problems at Google:

- multicore processors
- networked systems
- massive computation clusters
- scale: 10^7 lines of code
- scale: 10^3 programmers
- scale: 10^{6+} machines (design point)

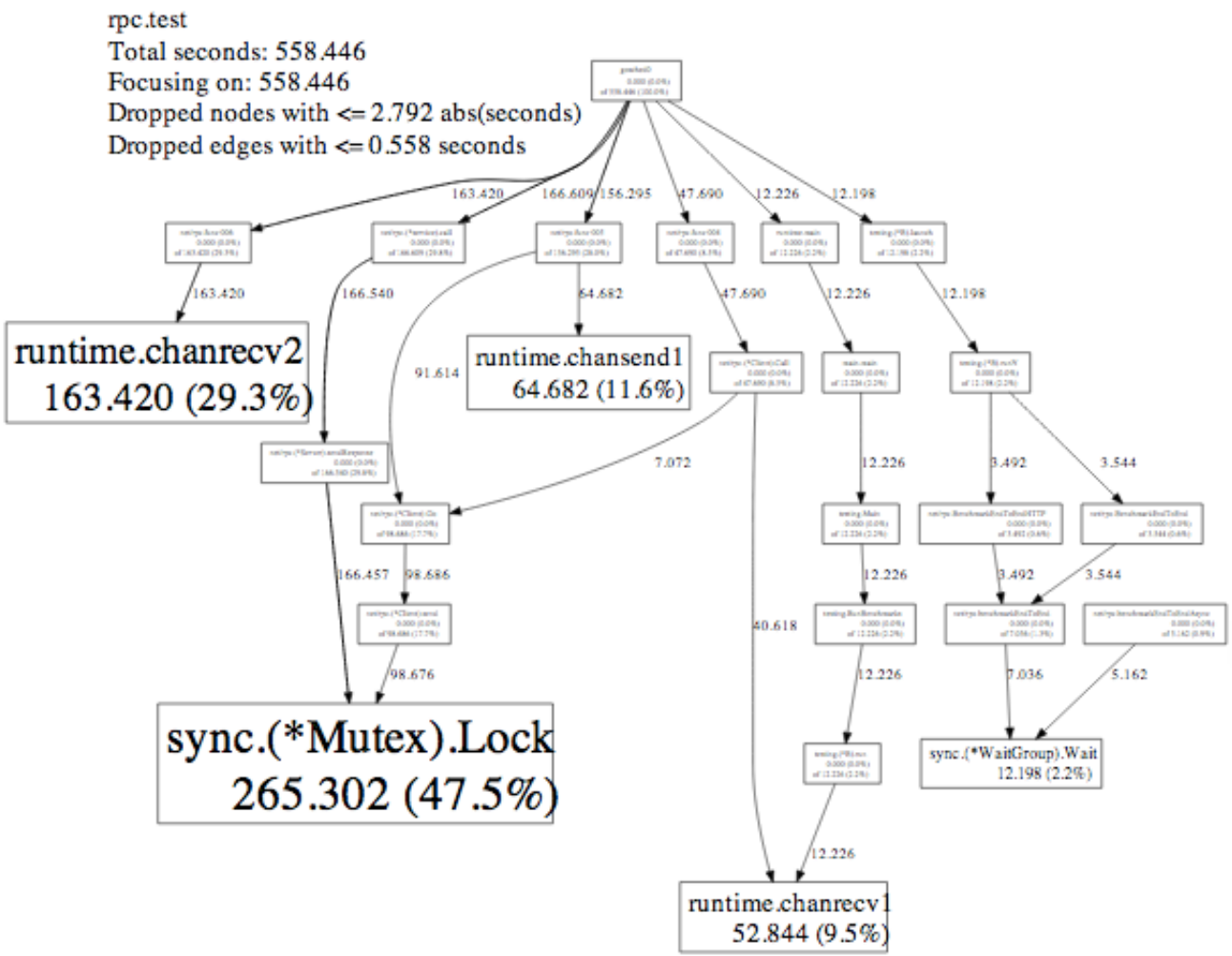
Deployed: parts of YouTube, dl.google.com, Blogger, Google Code, Google Fiber, ...

Great Tools

- go [build] [run]
- go get
- gofmt, goimports
- godoc
- go test [-cover] [-race]
- go vet
- gofix, gofmt -r, eg
- golint
- godep
- present

Profiling visualization (web)

```
$ go tool pprof
```



Coverage visualization

```
$ go tool cover -html=cover.out
```

strings/strings.go not tracked not covered covered

```
// isSeparator reports whether the rune could mark a word boundary.
// TODO: update when package unicode captures more of the properties.
func isSeparator(r rune) bool {
        // ASCII alphanumerics and underscore are not separators
        if r <= 0x7F {
                switch {
                case '0' <= r && r <= '9':
                        return false
                case 'a' <= r && r <= 'z':
                        return false
                case 'A' <= r && r <= 'Z':
                        return false
                case r == '_':
                        return false
                }
                return true
        }
        // Letters and digits are not separators
        if unicode.IsLetter(r) || unicode.IsDigit(r) {
                return false
        }
        // Otherwise, all we can do for now is treat spaces as separators.
        return unicode.IsSpace(r)
}
```

Hello World

```go
package main

import "fmt"

func main() {
    hello := "Hello, world."
    fmt.Println(hello)
    hello = "Hello, world, again."
    fmt.Println(hello)
}
```

Run

```
$>go run hello.go
Hello, world.
Hello, world, again.
```

Concurrency

Goroutines

A goroutine is a thread of control within the program, with its own local variables and stack. Cheap, easy to create.

- Concurrency is not parallelism, although it enables parallelism.
- A goroutine runs concurrently (but not necessarily in parallel).
- If you have only one processor, your program can still be concurrent but it cannot be parallel.

golang.org/s/concurrency-is-not-parallelism (http://golang.org/s/concurrency-is-not-parallelism)

You can dispatch millions of goroutines - and OS native threads NOT !

Without goroutines

```
const count int = 5

func even() {
    i := -1
    for {
        i++
        if i%2 == 0 {
            fmt.Printf("even is the dominant (%v)!\n", i)
            if i < count {
                runtime.Gosched()
            }
        }
    }
}

func odd() {
    i := 0
    for {
        i++
        if i%2 == 1 {
            fmt.Printf("odd is the dominant (%v)!\n", i)
            runtime.Gosched()
        }
    }
}
```

```
func main() {
    runtime.GOMAXPROCS(1) // only one OS native thread
    even()
    odd()
    time.Sleep(time.Second)
}
```

Run

Output:

```
$>go run goroutines101.1.go | head -20
even is the dominant (0)!
even is the dominant (2)!
even is the dominant (4)!
even is the dominant (6)!
even is the dominant (8)!
even is the dominant (10)!
even is the dominant (12)!
even is the dominant (14)!
even is the dominant (16)!
even is the dominant (18)!
even is the dominant (20)!
even is the dominant (22)!
even is the dominant (24)!
even is the dominant (26)!
even is the dominant (28)!
even is the dominant (30)!
even is the dominant (32)!
even is the dominant (34)!
even is the dominant (36)!
even is the dominant (38)!
```

With goroutines - single-thread

```
const count int = 5

func even() {
    i := -1
    for {
        i++
        if i%2 == 0 {
            fmt.Printf("even is the dominant (%v)!\n", i)
            if i < count {
                runtime.Gosched()
            }
        }
    }
}

func odd() {
    i := 0
    for {
        i++
        if i%2 == 1 {
            fmt.Printf("odd is the dominant (%v)!\n", i)
            runtime.Gosched()
        }
    }
}
```

```
func main() {
    runtime.GOMAXPROCS(1) // only one OS native thread
    go even()
    go odd()
    time.Sleep(time.Second)
}
```

Run

Output:

```
$>go run goroutines101.2.go | head -20
even is the dominant (0)!
odd is the dominant (1)!
even is the dominant (2)!
odd is the dominant (3)!
even is the dominant (4)!
odd is the dominant (5)!
even is the dominant (6)!
even is the dominant (8)!
even is the dominant (10)!
even is the dominant (12)!
even is the dominant (14)!
even is the dominant (16)!
even is the dominant (18)!
even is the dominant (20)!
even is the dominant (22)!
even is the dominant (24)!
even is the dominant (26)!
even is the dominant (28)!
even is the dominant (30)!
even is the dominant (32)!
```

With goroutines multi-threads

```
const count int = 5

func even() {
    i := -1
    for {
        i++
        if i%2 == 0 {
            fmt.Printf("even is the dominant (%v)!\n", i)
            if i < count {
                runtime.Gosched()
            }
        }
    }
}

func odd() {
    i := 0
    for {
        i++
        if i%2 == 1 {
            fmt.Printf("odd is the dominant (%v)!\n", i)
            runtime.Gosched()
        }
    }
}
```

```go
func main() {
    runtime.GOMAXPROCS(runtime.NumCPU()) // use all computer logical CPUs
    go even()
    go odd()
    time.Sleep(time.Second)
}
```

Run

Output:

```
$>go run goroutines101.3.go | head -20
odd is the dominant (1)!
even is the dominant (0)!
odd is the dominant (3)!
odd is the dominant (5)!
even is the dominant (2)!
odd is the dominant (7)!
even is the dominant (4)!
odd is the dominant (9)!
even is the dominant (6)!
odd is the dominant (11)!
even is the dominant (8)!
odd is the dominant (13)!
even is the dominant (10)!
odd is the dominant (15)!
even is the dominant (12)!
even is the dominant (14)!
odd is the dominant (17)!
even is the dominant (16)!
odd is the dominant (19)!
odd is the dominant (21)!
```

Race Detector

```
package main

import "runtime"

//obvious race O_o
var i int

func main() {
    runtime.GOMAXPROCS(runtime.NumCPU()) // use all computer logical CPUs

    go func() {
        for {
            i++
        }
    }()
    go func() {
        for {
            i--
        }
    }()

    for {
    }
}
```

Run

Race detector example

```
$ go run -race race.go
==================
WARNING: DATA RACE
Read by goroutine 6:
  main.main.func2()
    /Users/urishamay/sandbox/codemotion2015/race.go:10 +0x30

Previous write by goroutine 5:
  main.main.func1()
    /Users/urishamay/sandbox/codemotion2015/race.go:7 +0x4c

Goroutine 6 (running) created at:
  main.main()
    /Users/urishamay/sandbox/codemotion2015/race.go:11 +0x50

Goroutine 5 (finished) created at:
  main.main()
    /Users/urishamay/sandbox/codemotion2015/race.go:8 +0x38
==================
Found 1 data race(s)
exit status 66
```

Channels

Channels

Problem: Prime sieve

Problem specification from
Communicating Sequential Processes, by C. A. R. Hoare, 1978

"Problem: To print in ascending order all primes less than 10000. Use an array of processes, SIEVE, in which each process inputs a prime from its predecessor and prints it. The process then inputs an ascending stream of numbers from its predecessor and passes them on to its successor, suppressing any that are multiples of the original prime. "

Solution

Defined in the 1978 CSP paper.

Channels

Channel communication is the main method of synchronization between goroutines.

Don't communicate by sharing memory, share memory by communicating.

Construction:

```
ch1  := make(chan int)     // make unbuffered channel (synchronous). 0 capacity by default
ch2  := make(chan int, 10) // make buffered channel (asynchronous)
ch1 <- 1 // send data to channel
<- ch2   // receive data from channel
```

Let's build web crawler!

Web crawler: dummy HTTP requests

```
// Don't Try This at Home ;D
// const count int = 1000000
const count int = 1000

func main() {

    for i := 0; i < count; i++ {
        go DoGet("https://golang.org")
    }

    done := make(chan bool)
    <-done
}

func DoGet(url string) {
    resp, err := http.Get(url)
    if err != nil {
        // handle error
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    if err != nil {
        // handle error
```

```
    }

    fmt.Println(string(body))
}
```

Run

Web crawler: let's wait to workers

```
// Don't Try This at Home ;D
// const count int = 1000000
const count int = 1000

func main() {

    var done sync.WaitGroup
    done.Add(count)

    for i := 0; i < count; i++ {
        go DoGet("https://golang.org", &done)
    }

    // will wait till all the goroutines notify Done
    done.Wait()
}

func DoGet(url string, done *sync.WaitGroup) {
    defer done.Done()

    resp, err := http.Get(url)
    if err != nil {
        // handle error
    }
```

```
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    if err != nil {
        // handle error
    }

    fmt.Println(string(body))
}
```

Run

Web crawler: let's communicate the results to main dispatcher

```
// Don't Try This at Home ;D
// const count int = 1000000
const count int = 1000

func main() {

    var done sync.WaitGroup
    done.Add(count)

    responses := make(chan *MyResponse, count)

    for i := 0; i < count; i++ {
        go DoGet("https://golang.org", &done, responses)
    }

    // will wait till all the goroutines notify Done
    done.Wait()
    close(responses)

    for response := range responses {
        fmt.Println(response)
    }
}

type MyResponse struct {
```

```
    Body  string
    Error error
}

func DoGet(url string, done *sync.WaitGroup, reponseChannel chan<- *MyResponse) {
    defer done.Done()

    my := new(MyResponse)

    resp, err := http.Get(url)
    if err != nil {
        my.Error = err
        reponseChannel <- my
        return
    }

    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)

    my.Error = err
    my.Body = string(body)
    reponseChannel <- my
}
```

Run

Let's build http upload server!!

You can use it as a Log-Receiver/Images-Upload-Server etc...

```go
// global shared counter, used by all goroutines,
// should synchronized - using atomic.Add and not a mutex
var x int64

// request handler - each handler run in a dedicated goroutine
func upload(w http.ResponseWriter, r *http.Request) {
    out, err := os.Create("/tmp/" + strconv.FormatInt(atomic.AddInt64(&x, 1), 10))

    if err != nil {
        fmt.Println(err)
    }

    // synchronus heaven - get rid of my back caLLbACK HeLL!O_o
    io.Copy(out, r.Body)
}

func main() {
    runtime.GOMAXPROCS(runtime.NumCPU())
    http.HandleFunc("/upload", upload)
    http.ListenAndServe(":8080", nil)
}
```

Run

Resources

The Go Programming Language

golang.org (https://golang.org)

A Tour Of Go
tour.golang.org (https://tour.golang.org)

Go Playground
play.golang.org (https://play.golang.org)

The Go Blog
blog.golang.org (https://blog.golang.org)

Go wiki
github.com/golang/go/wiki (https://github.com/golang/go/wiki)

Thank you

Uri Shamay
Lead Developer, Juno
shamayuri@gmail.com (mailto:shamayuri@gmail.com)

(#ZgotmplZ) github.com/cmpxchg16 (https://github.com/cmpxchg16)

@cmpxchg16 (http://twitter.com/cmpxchg16)

(#ZgotmplZ) cmpxchg16.me (http://cmpxchg16.me)

Standing on the shoulders of giants