Go Data Concurrency Detection

Deadlock and Race Detectors
25 April 2020

Emil Sharifullin
Lead Software Engineer, SKB Kontur

Go Go Go

- Easy to learn
- Have brilliant concurrency paradigm
- If you have more than one core concurrency become parallelism

Too many circumstances to create a bunch of concurrency errors => **Go needs tools to prevent it**

Deadlocks

Deadlocks

In concurrent computing, a deadlock is a state in which each member of a group is waiting for another member, including itself, to take action, such as sending a message or more commonly releasing a lock

```
func main() {
    ch1 := make(chan int)
    ch2 := make(chan int)
    go func() {
        fmt.Println(<-ch1)
        ch2 <- 1
    }()
    fmt.Println(<-ch2)
    ch1 <- 1
}
```

Run

Deadlocks

Deadlocks can appear even in single threaded environment

```
func main() {
    ch := make(chan int)
    ch <- 1
    fmt.Println(<-ch)
}
```

Run

```
func main() {
    var m1 sync.Mutex
    m1.Lock()
    m1.Lock()
}
```

Run

Scheduler

Go scheduler operates with P's, M's and G's

- **G:** goroutine
- **M:** OS thread (machine) can execute at most one G at a time
- **P:** logical processor can hold at most one M at a time

At any time, there are at most GOMAXPROCS number of P

Deadlock Detector

Go runtime have a deadlock detector implemented in goroutines scheduler

```
4430  // Check for deadlock situation.
4431  // The check is based on number of running M's, if 0 -> deadlock.
4432  // sched.lock must be held.
4433  func checkdead() {
```

Go Runtime https://github.com/golang/go/blob/master/src/runtime/proc.go

(https://github.com/golang/go/blob/885099d1550dad8387013c8f35ad3d4ad9f17c66/src/runtime/proc.go#L4433-L4531)

Counting M's

Running M's is a basically difference between all M's and idled M's

```
4463      run := mcount() - sched.nmidle - sched.nmidlelocked - sched.nmsys
4464      if run > run0 {
4465          return
4466      }
```

And check for system purpose M

```
4449      // If we are not running under cgo, but we have an extra M then account
4450      // for it. (It is possible to have an extra M on Windows without cgo to
4451      // accommodate callbacks created by syscall.NewCallback. See issue #6751
4452      // for details.)
4453      var run0 int32
```

If any M is running -> **No deadlock**

Checking G's

Is it any G alive in runtime?

```
4474      for i := 0; i < len(allgs); i++ {
4475          gp := allgs[i]
4476          if isSystemGoroutine(gp, false) {
4477              continue
4478          }
4479          s := readgstatus(gp)
4480          switch s &^ _Gscan {
4481          case _Gwaiting,
4482              _Gpreempted:
4483              grunning++
```

If no G is running -> **Deadlock**

```
4493      if grunning == 0 { // possible if main goroutine calls runtime·Goexit()
4494          unlock(&sched.lock) // unlock so that GODEBUG=scheddetail=1 doesn't hang
4495          throw("no goroutines (main called runtime.Goexit) - deadlock!")
4496      }
```

Checking P's

Eventually check that any P have actions to take at some time

```
4522      for _, _p_ := range allp {
4523          if len(_p_.timers) > 0 {
4524              return
4525          }
4526      }
```

If some action schedulled -> **No deadlock**

Deadlock

If not returned before -> **Deadlock!**

```
4528      getg().m.throwing = -1 // do not dump full stacks
4529      unlock(&sched.lock)    // unlock so that GODEBUG=scheddetail=1 doesn't hang
4530      throw("all goroutines are asleep - deadlock!")
```

Additional Cases

Do not check if panicking

```
4445      if panicking > 0 {
4446          return
4447      }
```

Do not check if code builded as .so library or archive

```
4437      if islibrary || isarchive {
4438          return
4439      }
```

And additional P's timers case for code running in playground

```
4498      // Maybe jump time forward for playground.
4499      if faketime != 0 {
```

Not a Magic

```
func main() {
    ch1 := make(chan int)
    go func() {
        fmt.Println(<-ch1)
    }()
    sigs := make(chan os.Signal, 1)
    signal.Notify(sigs, syscall.SIGINT, syscall.SIGTERM)
    <-sigs
}
```

Run

Data Races

Data Races

A data race occurs when two threads access the same variable concurrently and at least one of the accesses is write.

```
counter := 0
var wg sync.WaitGroup
for i := 0; i < 10000; i++ {
    wg.Add(1)
    go func() {
        counter = counter + 1
        wg.Done()
    }()
}
wg.Wait()
fmt.Println(counter)
```

Run

Data Race Detector

Let's check it

```
#!/usr/bin/env bash

go run -race src/race/first/main.go
```

Run

It's a result of ThreadSanitizer algorithm

ThreadSanitizer – data race detection in practice by Serebryany & Iskhodzhanov

(https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35604.pdf)

Happens Before

Is a way to order events between different executors(processes/threads/goroutines) in concurrent(distributed) systems

Event **E1** happens before event **E2(E1<E2)** if at least one of following is true

- **E1** and **E2** in one executor and E1 placed earlier than E2
- **E1** and **E2** is a Lock-Unlock events of one synchronization primitive
- Exist **E'** that **E1<E'** and **E'<E2**

In the other cases **E1** and **E2** are concurrent

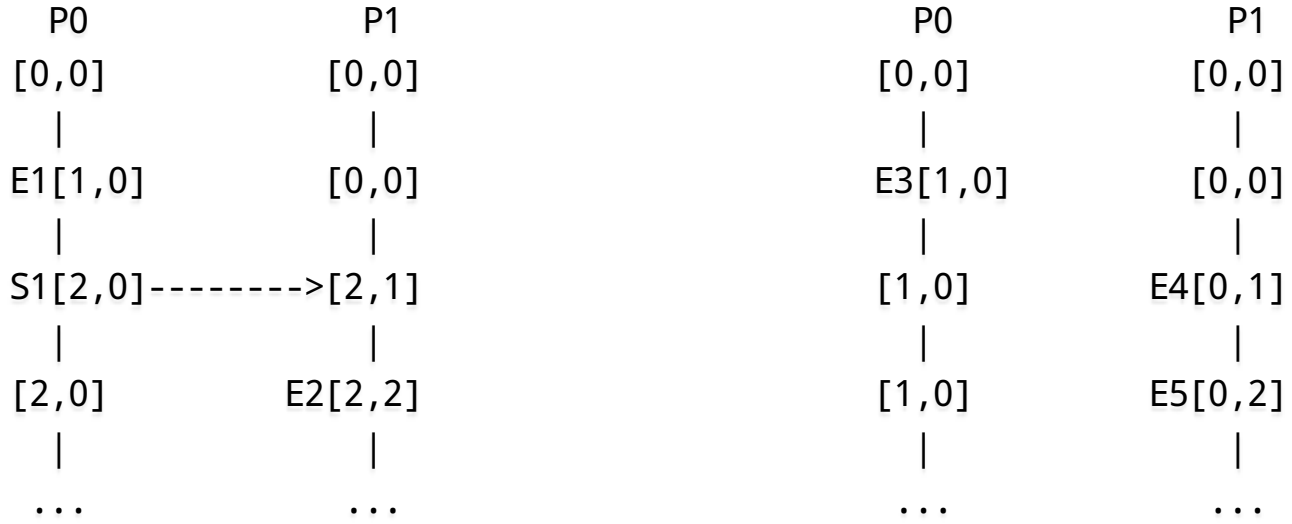
Vector Clocks

A vector clock is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations

It's based on Lamport clock idea and was developed by Colin Fidge and Friedemann Mattern

Allow us to determine Happens Before relation in distributed systems

Vector Clocks in Practice



So **E1 < E2** if **V[E1] < V[E2]**

V[E1] = [1,0], V[E2] = [2,1]

V[E1] < V[E2] => E1 < E2

V[E3] = [1,0], v[E4] = [0,1]

V[E3] !< V[E4] => E3 !< E4

V[E4] !< V[E3] => E4 !< E3 => Concurrent!

Applying to Go

All these concepts are implemented in Race Detector that called ThreadSanitizer

- **E1..En -** Memory reads and writes
- **S1..Sn** - Mutex Lock/Unlock & Goroutines creation

On E1..En we compare current event with previous events that affect this memory location and do following checks:

- Is at least one of events **write**?
- Can we identify **happens before** relation?

If answers **Yes** && **No** => **Data Race**

Implementation

```
182     0x001d 00029 (src/race/first/main.go:15)    PCDATA    $0, $1
183     0x001d 00029 (src/race/first/main.go:15)    PCDATA    $1, $1
184     0x001d 00029 (src/race/first/main.go:15)    MOVQ    "".&counter+32(SP), AX
185     0x0022 00034 (src/race/first/main.go:15)    PCDATA    $0, $0
186     0x0022 00034 (src/race/first/main.go:15)    INCQ    (AX)
```

```
297     0x0026 00038 (src/race/first/main.go:14)    CALL    runtime.racefuncenter(SB)
298     0x002b 00043 (src/race/first/main.go:15)    PCDATA    $0, $1
299     0x002b 00043 (src/race/first/main.go:15)    MOVQ    "".&counter+40(SP), AX
300     0x0030 00048 (src/race/first/main.go:15)    PCDATA    $0, $0
301     0x0030 00048 (src/race/first/main.go:15)    MOVQ    AX, (SP)
302     0x0034 00052 (src/race/first/main.go:15)    CALL    runtime.raceread(SB)
303     0x0039 00057 (src/race/first/main.go:15)    PCDATA    $0, $1
304     0x0039 00057 (src/race/first/main.go:15)    MOVQ    "".&counter+40(SP), AX
305     0x003e 00062 (src/race/first/main.go:15)    MOVQ    (AX), CX
306     0x0041 00065 (src/race/first/main.go:15)    MOVQ    CX, "".autotmp_7+16(SP)
307     0x0046 00070 (src/race/first/main.go:15)    PCDATA    $0, $0
308     0x0046 00070 (src/race/first/main.go:15)    MOVQ    AX, (SP)
309     0x004a 00074 (src/race/first/main.go:15)    CALL    runtime.racewrite(SB)
310     0x004f 00079 (src/race/first/main.go:15)    MOVQ    "".autotmp_7+16(SP), AX
311     0x0054 00084 (src/race/first/main.go:15)    INCQ    AX
312     0x0057 00087 (src/race/first/main.go:15)    PCDATA    $0, $2
313     0x0057 00087 (src/race/first/main.go:15)    PCDATA    $1, $1
314     0x0057 00087 (src/race/first/main.go:15)    MOVQ    "".&counter+40(SP), CX
```

Implementation

Notify race detector on a new goroutine creation and mutex Lock/Unlock

```
3563      if raceenabled {
3564          newg.racectx = racegostart(callerpc)
3565      }
```

func newproc1 https://github.com/golang/go/blob/master/src/runtime/proc.go

(https://github.com/golang/go/blob/885099d1550dad8387013c8f35ad3d4ad9f17c66/src/runtime/proc.go#L3563-L3565)

```
75          if race.Enabled {
76              race.Acquire(unsafe.Pointer(m))
77          }
```

func (m *Mutex) Lock https://github.com/golang/go/blob/master/src/sync/mutex.go

(https://github.com/golang/go/blob/885099d1550dad8387013c8f35ad3d4ad9f17c66/src/sync/mutex.go#L75-L77)

```
180     if race.Enabled {
181         _ = m.state
182         race.Release(unsafe.Pointer(m))
183     }
```

func (m *Mutex) Unlock https://github.com/golang/go/blob/master/src/sync/mutex.go

(https://github.com/golang/go/blob/885099d1550dad8387013c8f35ad3d4ad9f17c66/src/sync/mutex.go#L180-L183)

Fixed Example

```
    counter := 0
    var wg sync.WaitGroup
    var m sync.Mutex

    for i := 0; i < 10000; i++ {
        wg.Add(1)
        go func() {
            m.Lock()
            counter = counter + 1
            m.Unlock()
            wg.Done()
        }()
    }
    wg.Wait()
    fmt.Println(counter)
```

Run

```
#!/usr/bin/env bash

go run -race src/race/second/main.go
```

Run

Pros and Cons

- Do not have false positives
- Widely used and do not have obvious bugs

- Some races can be not found
- Memory consumption increases 5-10 times
- Execution time increases 5-15 times
- ThreadSanitizer is a C++ library so works only with CGo enabled

When and How to Use

Deadlock detector

- Anytime
- Anywhere

Data race detector

- In tests **`go test -race ./...`**
- During development **`go run -race main.go`**
- On special environment for race detection **`go build -race .`**

Thank you

Deadlock and Race Detectors
25 April 2020

Emil Sharifullin
Lead Software Engineer, SKB Kontur
iam@litleleprikon.me (mailto:iam@litleleprikon.me)
@litleleprikon (http://twitter.com/litleleprikon)