Concurrency Learnings

GoSF
18 November 2015

Naitik Shah
Engineer, Parse/Facebook

My Starting Point

- PHP: `curl_multi_exec`: php.net (http://php.net/manual/en/function.curl-multi-exec.php)
- Hack: `async / await`: docs.hhvm.com (http://docs.hhvm.com/manual/en/hack.async.php)
- JS: Callbacks, Promises

Blissful Life

- Concurrent, but Single Threaded of execution
- No worry about mutexes
- Limited worry about operation ordering
- No worry about atomic operations
- Nothing happens between two lines of code

n00b Life

- Started with `sync.Mutex`
- Sometimes `sync.RWMutex`
- `atomic.AddInt64` and `atomic.LoadInt64` for simple counters

Graduation

- `sync.Once`
- `sync.WaitGroup`
- `chan`
- `atomic.Value`
- `sync.Cond`

A `chan` pattern: background manager goroutine

- Batching: github.com/facebookgo/muster (http://github.com/facebookgo/muster)
- Connection Pool: github.com/facebookgo/rpool (https://github.com/facebookgo/rpool)
- `http.ConnState` tracking: github.com/facebookgo/httpdown (https://github.com/facebookgo/httpdown)

Problem: Keep Slow Work Out

- There is only **1** manager goroutine
- While it is busy, everyone else is waiting

Solution: Keep Slow Work Out

Push IO to Caller using Sentinel Value

```
var dialConnSentinel = sentinelCloser(1)

func (p *Pool) Acquire() (io.Closer, error) {
    r := make(chan io.Closer)
    p.acquire <- r
    c <- r
    if c == dialConnSentinel {
        return p.dial()
    }
    return c, nil
}

func (p *Pool) manager() {
    for {
        select {
        case r := <-p.acquire:
            if connAvailable {
                r <- conn
                return
            }
            r <- dialConnSentinel
        }
    }
```

Solution: Keep Slow Work Out

Push IO to worker goroutines

```
func (p *Pool) manage() {
    closers := make(chan io.Closer, closeBacklog)
    var closeWG sync.WaitGroup
    closeWG.Add(closeConcurrent)
    for i := 0; i < closeConcurrent; i++ {
        go func() {
            defer closeWG.Done()
            for c := range closers {
                p.CloseErrorHandler(c.Close())
            }
        }()
    }
    defer func() {
        close(closers)
        closeWG.Wait()
    }()

    for {
        select {
        case c := <-p.closeConn:
            closers <- c
        }
    }
```

Problem: Not Blocking After Close

`Use-after-free` for concurrency?

- Graph of Components that work Concurrently
- Stopping in Order
- Gracefully handle incorrect Order

What happens when methods are called after the component has been stopped?

- `close` channels to ensure a `panic: send on closed channel`
- Use `sync.Once` to make operations idempotent (specifically `Stop / Close`)

An atomic pattern: snapshots

- Use case: `net/rpc.Client` pool (across hosts)
- `atomic.Value` based solution
- `lock-free` in the happy hot path

Unclear if the additional machine performance gain is worth the human cognitive cost.

Problem: things may happen more than once

- rpc.Client interface
- Balances requests across hosts
- Snapshot holds a slice of real rpc.Client instances, 1 for each host
- Multiple goroutines concurrently use a rpc.Client instance
- Multiple goroutines concurrently experience rpc failures

Solution: things may happen more than once

- Happy Hot Path: pick a connection from the snapshot
- Report It: got `rpc.ErrShutdown`, tell manager goroutine

```
    // we may get multiple reports of an entry being down, so we guard
    // against doing things twice.
    if existing, isAlive := alive[entry.addr]; isAlive {
        // we got a report of a down entry after we already successfully
        // reconnected to it. don't throw away a good client.
        if existing != entry {
            continue
        }
    }
```

Problem: time

- Testing functionality related to time is hard
- Don't want to wait for real time to pass

Solution: mock clock

- A forked version of Ben Johnson's clock package:

```
https://github.com/benbjohnson/clock
https://github.com/facebookgo/clock
```

- Would be great to make this solution more robust and add features

Problem: waiting on N channels

- Zookeeper Watches
- https://godoc.org/github.com/samuel/go-zookeeper/zk

```
ChildrenW(path string) ([]string, *Stat, <-chan Event, error)

GetW(path string) ([]byte, *Stat, <-chan Event, error)
```

Solution: reflect.Select

```
[]reflect.SelectCase

reflect.Select()
```

- Keep the dynamic and static channels separate if possible

Other Considerations

- Useful Zero Values enabled by `sync.Once`
- Testing `Close` and correct cleanup

Recap

- More things to worry about, but more powerful
- Exploit multi-core hardware, even on a single request
- Use tools like the race detector
- I like it

Thank you

Naitik Shah
Engineer, Parse/Facebook
naitik@fb.com (mailto:naitik@fb.com)