Debugging Go Programs

Tom Arrell

Thursday 28th May – Golang Meetup – SumUp

About me

Tom Arrell

- Senior Backend Engineer @ SumUp
- Logistics Squad
- twitter: twitter.com/tom_arrell
- github: github.com/tomarrell

Agenda

- Go at SumUp

Agenda

- Go at SumUp
- Go in Logistics

Agenda

- Go at SumUp
- Go in Logistics
- Why we're moving to Go

Agenda

- Go at SumUp
- Go in Logistics
- Why we're moving to Go
- Debugging...

Agenda

- Go at SumUp
- Go in Logistics
- Why we're moving to Go
- Debugging...
  - Primitive

Agenda

- Go at SumUp
- Go in Logistics
- Why we're moving to Go
- Debugging...
  - Primitive
  - GDB

Agenda

- Go at SumUp
- Go in Logistics
- Why we're moving to Go
- Debugging...
  - Primitive
  - GDB
  - Delve

Agenda

- Go at SumUp
- Go in Logistics
- Why we're moving to Go
- Debugging...
  - Primitive
  - GDB
  - Delve
  - Scenarios...

Go at SumUp

- Adopted within the last 2 years
- Mainly used for tooling
- First services written ~1 year ago
- Now migrating to Go for new services
- Deployed to Kube

Go in Logistics

- Joined around ~6 months ago
- Legacy Python scripts
  - Lack of monitoring
  - Email alerts
  - Git clone deployment
- Replaced with Go services
  - Deployed to Kubernetes
  - Prometheus, Sentry, OpsGenie
- 2 engineers with no prior Go experience brought up to speed

Go in Logistics

parser_error@sumup.com Mon, Dec 16, 2019, 10:45 PM

to alper.yildirim, vadym.dolinin, bernard.bedynski, caja.schorer, felix.jung, me

```
Traceback (most recent call last):
  File "/home/sumup/logistics-squad/dwh-logistics/responseParser/chile/chile_parser.py", line 83, in generate_upsert_values
    df_['order_item'] = df_.groupby(['sumup_order_id']).cumcount()+1
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/generic.py", line 3778, in groupby
    **kwargs)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/groupby.py", line 1427, in groupby
    return klass(obj, by, **kwds)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/groupby.py", line 354, in __init__
    mutated=self.mutated)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/groupby.py", line 2383, in _get_grouper
    in_axis, name, gpr = True, gpr, obj[gpr]
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/frame.py", line 1997, in __getitem__
    return self._getitem_column(key)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/frame.py", line 2004, in _getitem_column
    return self._get_item_cache(key)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/generic.py", line 1350, in _get_item_cache
    values = self._data.get(item)
  File "/usr/local/lib/python2.7/dist-packages/pandas/core/internals.py", line 3290, in get
    loc = self.items.get_loc(item)
  File "/usr/local/lib/python2.7/dist-packages/pandas/indexes/base.py", line 1947, in get_loc
    return self._engine.get_loc(self._maybe_cast_indexer(key))
  File "pandas/index.pyx", line 137, in pandas.index.IndexEngine.get_loc (pandas/index.c:4154)
  File "pandas/index.pyx", line 159, in pandas.index.IndexEngine.get_loc (pandas/index.c:4018)
  File "pandas/hashtable.pyx", line 675, in pandas.hashtable.PyObjectHashTable.get_item (pandas/hashtable.c:12368)
  File "pandas/hashtable.pyx", line 683, in pandas.hashtable.PyObjectHashTable.get_item (pandas/hashtable.c:12322)
KeyError: 'sumup_order_id'
```

Figure 1: Alerting, the old way

Go in Logistics

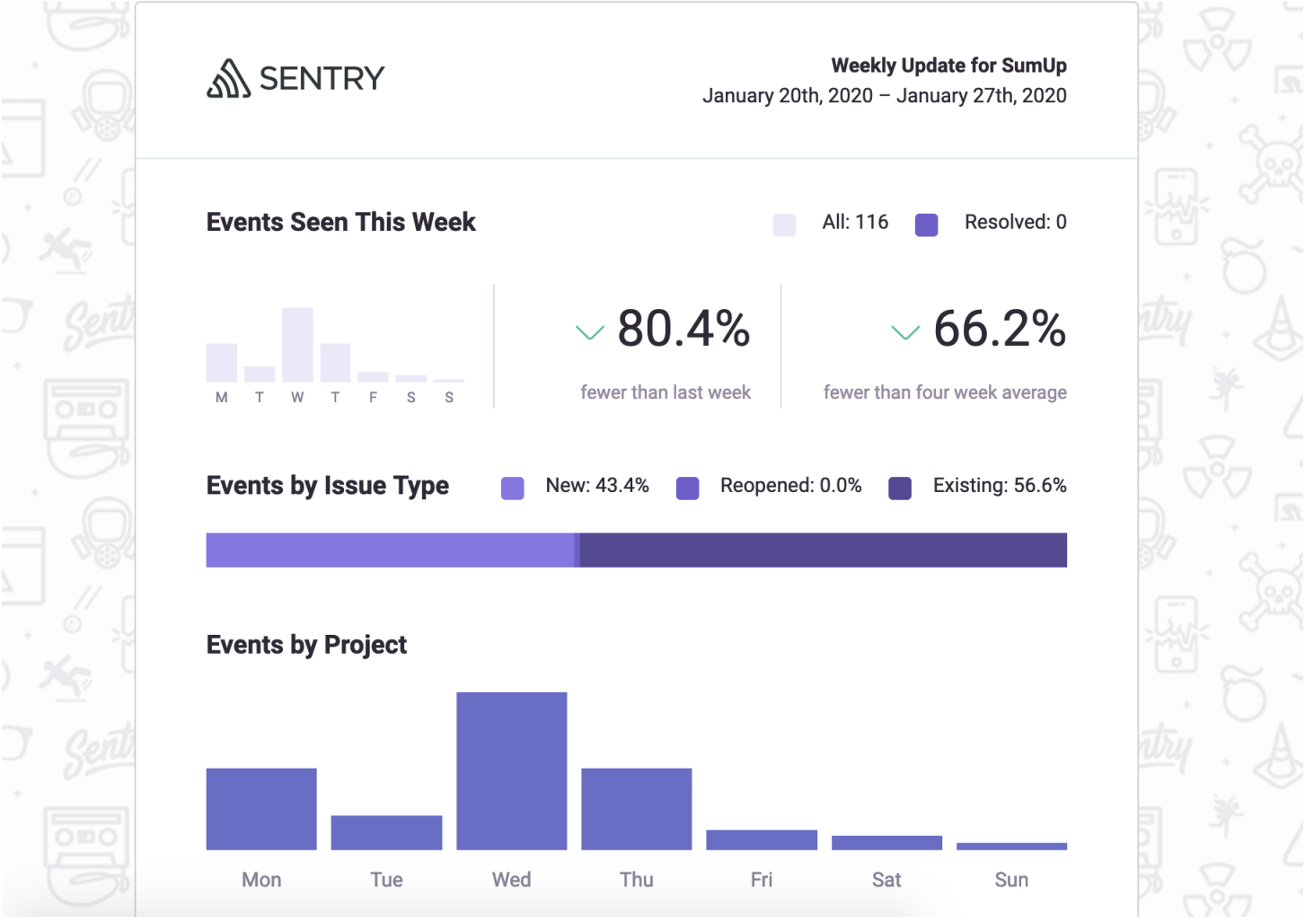

Figure 2: Alerting, the new way

Go in Logistics

Figure 3: Monitoring

Leaky Abstractions

All non-trivial abstractions, to some degree, are leaky.
– ***Joel Spolsky***

Leaky Abstractions

```rust
impl<TBehaviour, TInEvent, TOutEvent, THandler, THandlerErr, TConnInfo>
    ExpandedSwarm<TBehaviour, TInEvent, TOutEvent, THandler, THandlerErr, TConnInfo>
where TBehaviour: NetworkBehaviour<ProtocolsHandler = THandler>,
      TInEvent: Send + 'static,
      TOutEvent: Send + 'static,
      TConnInfo: ConnectionInfo<PeerId = PeerId> + fmt::Debug + Clone + Send + 'static,
      THandlerErr: error::Error + Send + 'static,
      THandler: IntoProtocolsHandler + Send + 'static,
      THandler::Handler: ProtocolsHandler<InEvent = TInEvent, OutEvent = TOutEvent, Error = THandlerErr>,
{
    /// Builds a new `Swarm`.
    pub fn new<TTransport, TMuxer>(transport: TTransport, behaviour: TBehaviour, local_peer_id: PeerId) -> Self
    where
        TMuxer: StreamMuxer + Send + Sync + 'static,
        TMuxer::OutboundSubstream: Send + 'static,
        <TMuxer as StreamMuxer>::OutboundSubstream: Send + 'static,
        <TMuxer as StreamMuxer>::Substream: Send + 'static,
        TTransport: Transport<Output = (TConnInfo, TMuxer)> + Clone + Send + Sync + 'static,
        TTransport::Error: Send + Sync + 'static,
        TTransport::Listener: Send + 'static,
        TTransport::ListenerUpgrade: Send + 'static,
        TTransport::Dial: Send + 'static,
    {
        SwarmBuilder::new(transport, behaviour, local_peer_id)
            .build()
    }
```

Figure 4: (Probably) leaky Rust Abstraction

Ruby to Go

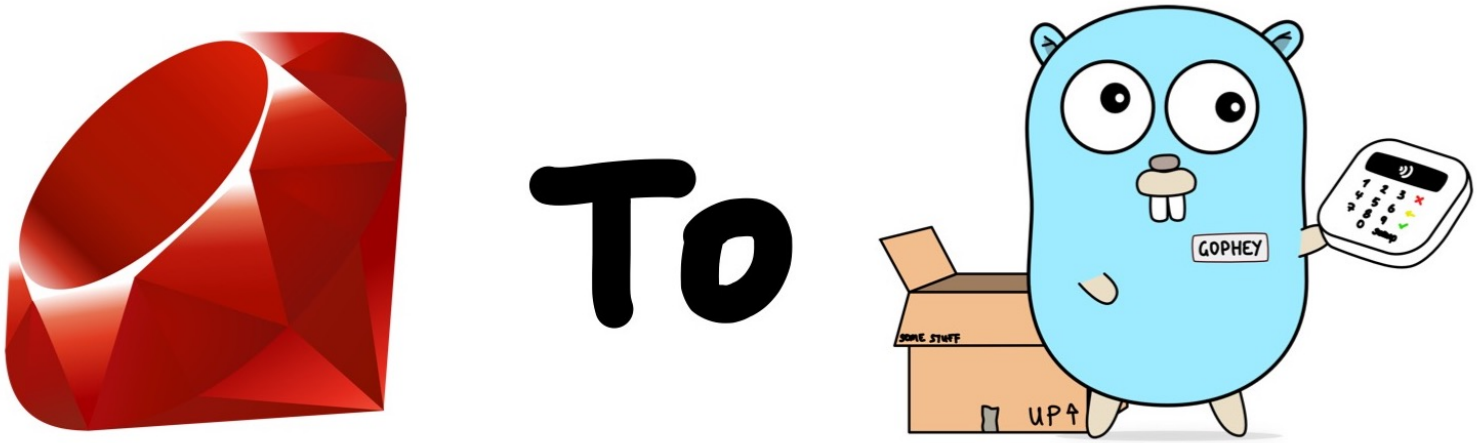

Figure 5: Ruby to Go, why?

Now what you came for... Debugging.

We'll take a look at a few contrived scenarios, and how we might be able to get some more insight with as little (or as much) effort as possible.

Words of Wisdom

If you dive into the bug, you tend to fix the local issue in the code, but if you think about the bug first, how the bug came to be, you often find and correct a higher-level problem in the code that will improve the design and prevent further bugs.

– ***Rob Pike***

fmt.Println()

fmt.Println() is the most universal, and all powerful debugger. Fight me.
– ***Me, circ. now***

fmt.Println()

```
package main

import "fmt"

func main() {
  fmt.Println("HERE")
  go func() {
    fmt.Println("Why are you not running?!")
  }()
  fmt.Println("HERE 2")
}
```

GNU Debugger

- Ok if you're using CGO
- Not so ok if you're writing plain Go
  - Defer statements
  - The scheduler, context switching
  - Custom type defs of builtin types
  - Some identifiers

Delve

- Dedicated debugger for Go programs
- Supports debugging:
  - Running processes
  - Examining core dumps
  - Built from scratch programs
  - Tests
  - Tracing

Scenario #1: Race Conditions

. . . ignoring this prohibition [of data races] introduces a practical risk of future miscompilation of the program.
– ***Hans-J. Boehm***

Scenario #1: Race Conditions

No race is a safe race.
– ***Me, just now***

Scenario #1: Race Conditions

Build your program with the `-race` flag.

Good idea to run your tests with this flag enabled.

Warning: The race detector is not infallible, and it may possibly miss certains cases. However, it will never report false positives.

Also.

```
$ go test -race mypkg   // check for races during tests
$ go build -race mycmd  // build a binary with R.D.
$ go run -race mysrc.go // immediate run with R.D.
```

Scenario #2: Deferred functions

Are you getting values back from your function that you don't expect?

Do you want to know which defer statements are being called?

Note:
The Go objdump tool displays the x86 assembly in **AT&T** syntax, whereas Delve displays it in **Intel** syntax.

Terms:

- **SP**: Stack pointer: top of stack.

Scenario #3: Post-mortem

Sometimes our application has already crashed and we'd like to get a better idea about the root cause.

One possible tool in our investigative toolbox are core dumps.

Setup:

```
ulimit -c unlimited # Remove core dump size limit
```

Terms:

- **Core dump**: A memory snapshot of a process, usually after a crash.

Scenario #4: Memory Leaks

Slightly more nuanced, may only become a problem over time.

- Prometheus client, exposes heap information
  - Use for heap size monitoring over time
- pprof
  - Use for heap inspection of running process to find problem objs

Heap profiling with `pprof`.

```
go tool pprof localhost:8080/debug/pprof/heap
> web
> top
```

Scenario #5: Goroutine Deadlocks

Very hard to debug.

pprof **blocking** profile not useful for deadlocked routines, but useful for finding contentious resources in your program.

pprof lets you inspect the state of each goroutine in your program.

To get source annotated view of all goroutines in package `pkg`.

```
go tool pprof localhost:8080/debug/pprof/goroutine
> list [pkg]
```

Alternatively, you can look at the **full goroutine stack dump** using:

```
curl localhost:8080/debug/pprof/goroutine?debug=2
```

Delve Tips

- Debug your test binaries as well
- `runtime.Breakpoint()`
- Checkpoints let you restart the program from a specific point
  - *Linux only*

"Use the right tool for the job."
*– **Someone**, I'm pretty sure*

–

fin

Questions?