Why are there Go programmers?

3 Apr 2013

Blake Mizerany
Engineer, Heroku

I work at:

What is Go?

- A new, open-source programming language
- Concurrent, garbage-collected, builds fast at scale.

Created by

- An impressive team of Google Engineers

Unix, UTF-8, Plan9, Memcache, RE2 (Regexp engine), and much more.

- More than 250 non-Google contributers, including myself.

Go 1

Released in March 2012

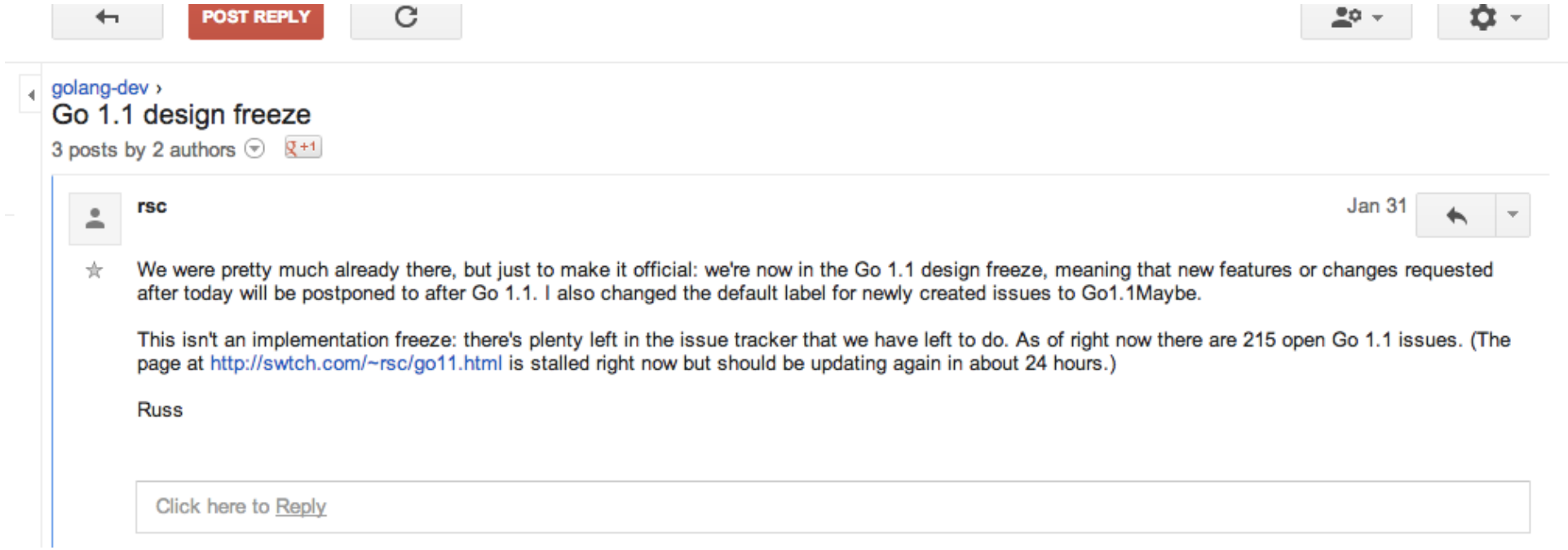
A specification of the language and libraries that will be supported for years.

The guarantee: code written for Go 1.0 will build and run with Go 1.x.

They won't break the spec or library APIs from 1.8 to 1.9 (you know who you are).

Go 1.1

Imminent



- Faster, faster, and faster

Hello, World

Run

What I love about Go (alphabetical order):

- Benchmarking tools
- Concurrency primitives
- Documentation
- Fast as a feature
- Functional Aspects
- Interfaces
- Profiling tools
- Simplicity
- Standard Library
- Statically linked binaries
- Testing tools

I use go to build (alphabetical order):

- Command line tools - Simple to not so simple
- Computationally heavy systems
- Network services
- Systems in need of efficient memory usage
- Web sites of all sorts

(This presentation software is a Go program!)

godoc.org/code.google.com/p/go.talks/pkg/present (http://godoc.org/code.google.com/p/go.talks/pkg/present)

Standard Libraries

- encoding/json (and XML for you "enterprise" people)
- http (the most comprehensive http library, ever - Server & Client)
- exec
- and so much more (plenty of references online)

go Tool - Building

Zero configuration (no Makefiles, or the like)

Build an executable

Run a single file program:

Install the current directories package

Install a remote package

go Tool - Utils

Format go code (no more boring arguments):

Analyze programs:

Got it!

Let's see some real code.

Problem

I need to compute quantiles for data sets with billions of data points over a stream; Way more than I can/want to keep in memory.

Effective Computation of Biased Quantiles over Data Streams

www.cs.rutgers.edu/~muthu/bquant.pdf (http://www.cs.rutgers.edu/~muthu/bquant.pdf)

Thank you: Graham Cormode, Flip Korn, S. Muthukrishnan, Divesh Srivastava

Enables list compression because:

"The algorithm keeps information about particular items from the input, and also stores some additional tracking information."

Perks

NOTE: The code for this package has change drastically since this presentation.

A pure Go library that implements ECBQDS.
It will compute biased quantiles for **Billions** of data points with little more than 4k of memory per stream, quickly.

The type

Initializer

Insert

mergeFunc - Higher-order functions

Query

Non-streaming example

Benchmark

Optimize - Buffer

We can batch inserts:

Benchmark Buffer

Digression: net/http

Go's net/http library is the most comprehensive HTTP library.

"I want Go's HTTP library to be the most complete HTTP library."
- Brad Fitzpatrick, Go core team member and maintainer of net/http.

Example

Run

Interfaces - Handlers

HandlerFunc

What is net/http.Server doing?

A few things:

- each invocation of serveHello runs in its own goroutine
- Keep-Alive is honored (idle connections are kept in a pool for later use)

goroutines

- Goroutines are lightweight threads that start around 4k in stack size.
- grow and shrink as needed
- require a few machine calls to start/grow/shrink
- very cheap!
- have millions of them for all you care!

Back to streaming quantiles

Right. We want to **stream** quantiles. How about HTTP?

The Request

Problem

If each invocation of a http.Handler is in it's own goroutine, we need our Buffer data-structure to be thread-safe.

- Composite sync.Mutex
- Composite *quantile.Buffer

serveInput

Problem

We're not using all of our CPUs efficiently because each insert is blocking the request with Lock.
I would like to respond ASAP to the client.

Channels

Channels are built-in communication primitives.

Unbuffered

Buffered

Channels send/recv

Send

Receive

Using channels

Send them data

Flushing to DB using select

Distributed Query

Later, when we want to query, we retrieve all the sample sets written for the second, merge them, and call q.Query()

We have solved THE Big Data problem

- Tested
- Benchmarked
- Used as much machine power as we can muster
- Thread-safe
- Very little code
- Proper error checking

All together

Thank you

3 Apr 2013

Blake Mizerany
Engineer, Heroku

(mailto:blake@heroku.com)

(http://heroku.com/)

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