www.m5.net
www.shoretel.com

franko@m5.net | architect

Who are we?

We provide phone systems and applications with an experience that businesses love.

So what does that mean?

- Hosted business phone systems
- VoIP: Just a phone and an Internet connection client side
- Thousands of companies hosted on our servers
- Brilliantly simple (so marketing tells us)
- Recently acquired by Shoretel, who make premise based systems

Our cloud legacy platform

- C/C++ based
- Massively parallel
- Scalable
- Robust
- Used by many tens of thousands of people concurrently

Why change?

Because its actually quite hard, and we need better tools

Really? What's so hard?

- One bad pointer can disrupt thousands of people
- Upgrades are challenging - hard to hot upgrade C. Downtime is bad, ok?
- Debugging stack trace for a pointer to a template class in gdb. Ouch.
- C threads are expensive, and synchronization challenging

Why are Linux C threads so painful for us?

- Thousands upon thousands of long lived (in weeks or months) busy connections, one or more per phone

- Thread per connection would be nice...

- Complex state machine, stored session data, and scheduler sharing few OS threads amount 10's of thousands of 'sessions'

- Complicated code that's easy to get wrong and difficult to debug

Spend most time in the details

Should really be spending most time architecting

Why Erlang?

It's supposed to be good for phones, or something. Everyone's seen the movie, right?

The marketing tells us...

- Stupidly parallel (more processes than atoms in universe!)
- Hot code loading
- Simple syntax, more meaningful code
- Crash isolation
- Easy to test
- More fun than Disneyland

Framework for distribution

- Built in multi node IPC
- Startup and supervision trees
- Mnesia distributed database
- All things we'd built ourselves, but here they were more featured and more mature
- A language and programming paradigm designed ground up for robustness and stability

Did they lie?

No more than marketing usually does. We'll get to that, but first:

What we did

Read the book, write some code. Can't be that hard, can it?

First contact: TFTP project

- Australian Linux Conference, 2007
- Our first convert returns, raving (mad, we thought)
- Sounded too good to be true
- Convinced us to give it a try in an experimental project: TFTP server to serve up dynamic configuration
- Success, on a small scale (failures due more to TFTP protocol on WAN than Erlang)

PHLEM: Phone Logic Emulator

- Another test to test other stuff. Needed a tool to test the system as a whole - a phone emulator.

- Started with just a single process

- Half a day turned it in to hundreds of emulated phones

- Another half day to turn it in to thousands of phones spread across many machines.

- Erlang UDP not quite good enough; C layer via Erlang port

CTIN ('kitten') project: Client Telephony Interface

- Product folk wanted 'Business Intelligence'. The sort of data they wanted was not in the original core

- Hard to rework existing C core to provide the sort of detail we needed

- Complex logic and data analysis on unordered events: Easier in Erlang than C

- Erlang part only a few thousand lines

Caller Name Lookup Project

- The start of our 'good' Erlang. Very little code, the time to implement was short.

- Debugged someone else's code during production emergency in 30 minutes. And patched.

- Lesson in supervision trees: get them right. We caused the whole application to shut down.

- Since the first failure; it just works.

Hosted call recording

- Major new feature for the system; lots of careful design, fast to implement. Focused on design and architecture; not details!

- C layer for media forking/recording

- Erlang layer for storing, retrieving encoded media via simple HTTP API

- Calls made up of many segments, so Erlang layer controlling mixing streams for recorded audio.

- Very robust and stable. Uptime of 6 months for some modules

Everything new is now Erlang

- New SIP redirect server (using YXA stack)

- Replaced data cache layer and db sync with Erlang version. Optimized, smarter version: easier to do in Erlang

- Working on new Shoretel phone support: new stack and user layer in Erlang, with old C core event bridge

- Enhancing testing tools

What we learned, mistakes we made

Boy, did we do a lot of both!

VM can crash

- Out of memory
- Non tail recursive loops
- Queue overflow
- Selective receive
- Consider running separate VM's for critical processes or 'edge' applications that receive 3rd party input

Erlang as a UNIX service

- Erl -noshell -detached : always returns success!
- What happen if startup fails?
- Says admins hate this
- Log rotation: no way to catch SIGHUP
- Erld: github.com/ShoreTel-Inc/erld

Hot code loading

- It's great; but still hard
- Patchy documentation
- No good tools to help
- Doesn't integrate with package management systems (eg. rpm)
- Good for little things, hard for the big things

Erlang tools and libraries

- System monitoring: lots of useful tools; but take the time to find and learn them BEFORE you have a problem

- OTP: designed around robust behavior. Rule of thumb: if in doubt, use it, unless you really know better

- Dialyzer: your best friend. Really.

Databases (I know we should be using Riak, but...)

- Mnesia: great moderate sized data sets, but:
- building indexes can be very slow - startup time problem
- SQL and ODBC - a little flakey and slow

Want to know more?

Erlang in production: "I wish I'd known that when I started" - Bernard Duggan

Linux conf australia 2012
http://www.youtube.com/watch?v=G0eBDWigORY

So you want to start using Erlang?

Here's our advice

Start small

- Start with things not core to your system!

- One piece at a time: you won't convince anyone that you have to take a year off to rewrite the system ground up in a new language

- Don't try do anything critical as your first project; you've got too much to learn

- Erlang is easy, but building concurrent, high load, reliable production systems will still require care

Testing tools

- It's a natural fit
- Outside your existing code base, so easy to write and low risk
- Easy to write concurrent version: great for load testing!
- Great way to get other people interested

Identify small self contained modules or jobs

- Like testing tools, these are often small, and isolated from the rest of your system

- Cron style jobs, db cleanup/maintenance, email notification scripts, monitoring tools, etc

- Any task you might otherwise use a scripting language, consider writing in Erlang as a learning exercise

- Gradually move on to larger modules as you gain experience

APIs

- You've already got them - use them as a natural process boundary

- Erlang is good at binary pattern matching - interpret c structures

- Code gen - create erlang records and c structs using something like erlydtl

- Ports to external c apps for code too tweak-ey to rewrite

Learn!

- It's easy, but it's also a whole new world of mistakes to make

- Consult the mailing list - everyone is friendly

- There's actually several books now- read them; they're all excellent

- OTP: bite the bullet early, and learn it. It's easier than kicking yourself later for all that duplicated effort

Market it

- Marketing? Are you serious? "Dammit, Jim, I'm an engineer, not a sales drone!"

- It's more important than you realize

- Get other people excited

- You've won when the CEO asks you how that erl-wotsit is going

Erlang does not absolve you of the responsibility to think

You can now write bad code faster than ever!

Most importantly

Have fun

One more thing...

We're recruiting. Check the flyer in your grab bag; or

www.m5.net
www.shoretel.com
franko@m5.net