Developing Erlang At Yahoo

Nick Gerakines and Mark Zweifel

Lots of "Official" Languages

- C/C++
- Java
- PHP
- Perl

Unofficial Languages

- Ruby
- Python
- Objective-C
- ... Erlang!

Not the first to go down this path.

Delicious

2.0 Launch is Huge

- Out on July 31st -- Over a year in the making
- Complete rewrite, front to back

Uses Erlang!

- Mostly C++ (is OO, I know)
- Ties to several subsystems to delegate large tasks, aka spam, search, algo, etc
- Several subsystems built in Erlang

Use Case #1

Data Migrations

Rewrites are hard.

More than just a row-to-row data copy.

Not just one.

2.0 involved simultaneous front-end and back-end development

There were several migrations of the entire system done over the course of development

First Attempt

- Written in Perl
  - Multiple threading models used
- No throttling or scaling of work in real-time
- Hard to debug
- Start/Stop was a nightmare

~~Second~~ Attempt
^ last

- Rewritten into Erlang services
- Crazy-fast
- System was introspective and self-monitoring
  - Dynamic scaling/throttling
  - Live migration status updates

Compute, Store & Write

- Created large snapshots of the entire d1 system for processing
- Phase 1 -- Compute diffs and store
  - Fragmented Mnesia stores around ~50 gigs a piece, up to 6 “cells”
- Phase 2 -- Write data into d2 system

Caveats

- Reading from a static data store (stopped slave).
- Activatee load tests and QA work.
- Multiple backends for migrations, stress test /QA environment and a “gold” production image.

Concurrency saved migrations

Ports!

- Several systems required interfaces to Perl scripts or C/C++ libraries
- Leveraged data auditing tool in Perl
- Could recycle non-Erlang code to really maximize efficiency
- Included Yahoo! specific functions, string/language encoding and detection.

Use Case #2

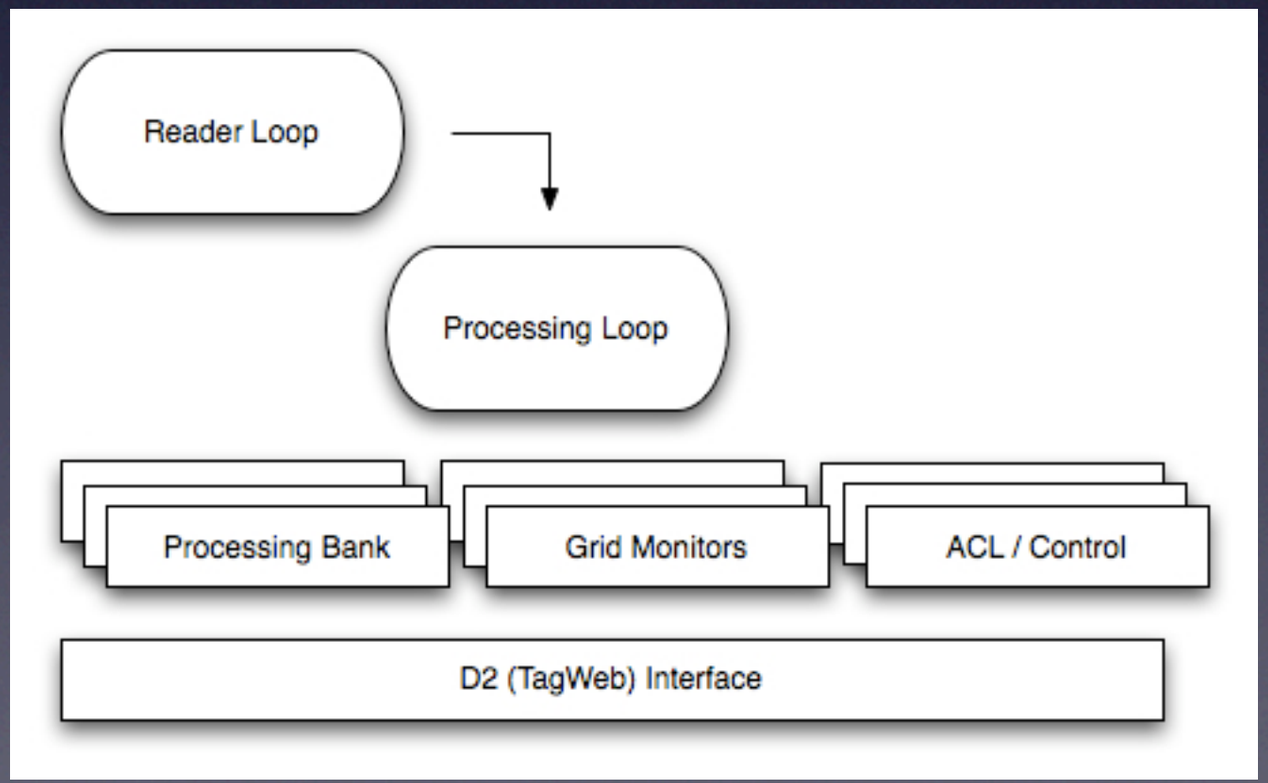
Rolling Migrations

There was no before

This entire system was written in Erlang from scratch
to bring the entire d2 system up to date to the second.

Architecture

- d1 Reader loop -- Monitors changes in the d1 system
- d1 Processing loop -- Would act on the changes and prepare them for d2 input



Erlang/OTP

Mnesia

Yeah, that's it.

Use Case #3

Algorithmics

Before

- Perl on top of cron jobs
- Perl can be difficult to manage
- Jobs can be very database intensive

After

- Rewritten into a number of small, independent systems
- Systems can be tweaked while live and running in production
- No cron, all running in real time
- Self-monitoring recursive operations

Concurrency

- Could leverage 600-700% of the CPU
- Computations were made friendly to parallel processing
- Introspection facilities let us scale up and down load to run at peak throughput

Erlang/OTP

Mnesia

Sound familiar?

Delicious

Complications

“If we knew what we were doing, it wouldn't be called research, would it?”
-- Albert Einstein

- Erlang is very different.
- Engineers are usually stubborn.
- Not enough critical mass at Yahoo to be unquestionable.
- Tension was already high, adding a new language into the mix added uncertainty.

Using Erlang At Yahoo

Strengths

- Extremely good at fault-tolerant distributed applications.
- Ideal for messaging, communications and logging.
- Long running jobs with heavy monitoring requirements.
- Agile development process
- Many different interfaces
- RPC baked in for free

Weaknesses

- There are documentation gaps.
- Hasn't achieved critical mass yet.
- The community is thin.

What We Did

- Internal packages and builds for multiple platforms.
- Created a simple build process based on a single Erlang install path.
- Standardized start/stop processes.

Thanks

Nick Gerakines <gerakine@yahoo-inc.com>
Mark Zweifel <markez@yahoo-inc.com>
Yogish Baliga <baliga@yahoo-inc.com>
Chris Goffinet <goffinet@yahoo-inc.com>