Lessons learned

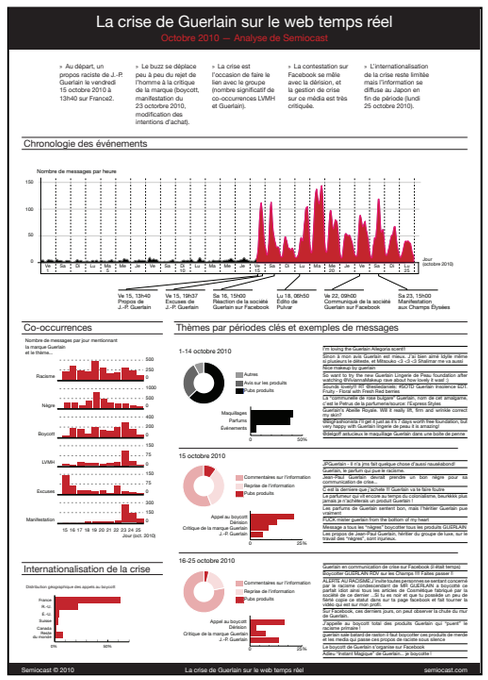
How we use Erlang to analyze millions of messages per day

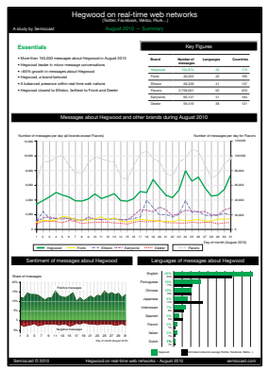
A few words about us

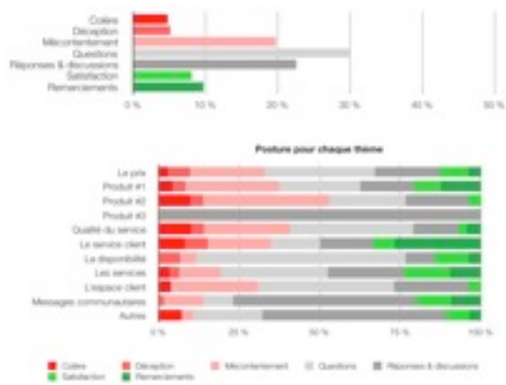
What we do with Erlang

A few words about us

Semiocast processes social media conversations to provide analytics and market research insights







Semiocast's offer

Quantitative studies
ad hoc

- Barometers
- Shares of social media conversations
- Sentiment analysis and clustering
- Topic identification
- Ad hoc quantitative indicators

Qualitative studies
ad hoc

- Consumer insights
- Verbatim research
- Clustering of conversations
- Mapping of communities and influence analysis

Monitoring

- Enumeration of social media conversation spaces
- Real-time alerts
- Daily/Weekly/Monthly reports
- Crisis monitoring

Tools

- Social media monitoring platform (Semioboard)
- Technology as a service (API)

Semioboard

Semioboard

Make sense of social media conversations

Quick demo

L'Oréal

by Semiocast

Semioboard

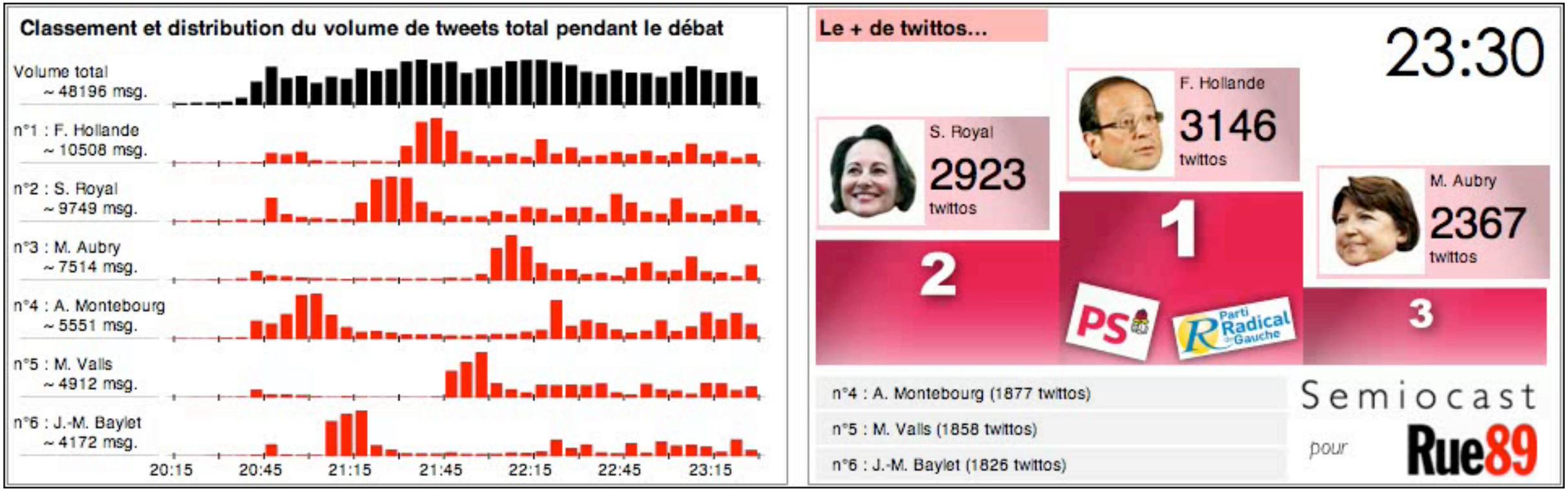
Semioboard

Make sense of social media conversations

feature demo:
smart buzz tagging

by Semiocast

Live analysis of comments on TV debates



How we ended up using Erlang

How we ended up using Erlang

Discovered Erlang when getting WiFi rabbits to talk to each other over XMPP (ejabberd) in 2007

Taught OCaml in 2004

Three reasons why we chose Erlang :

- hot code change and inspection
- fault-tolerance
- happy to do functional programming (gave us a break from Java and C++)

How we use Erlang

1352 OTP **releases**

- 47 applications
- 100K lines of Erlang (without tests)
- 11K lines of C/C++ (mostly glue)
- 1K lines of Java (glue)

~50 **ungraduated applications** for

- prototyping
- short lived projects
- web-based/command line tools that run on dev machines

```
{release, {"Semiocast OTP", "1352"}, {erts, "5.8.4"},
    [
        % erts 5.8.4
        {kernel, "2.14.4"},
        {stdlib, "1.17.4"},
        {mnesia, "4.4.18"},
        {inets, "5.5.2"},
        {sasl, "2.1.9.3"},
        {crypto, "2.0.2.1"},
        {snmp, "4.19"},
        {otp_mibs, "1.0.6"},
        {ssl, "4.1.5"},
        {public_key, "0.12"},
        {xmerl, "1.2.8"},
        {compiler, "4.7.3"},
        {runtime_tools, "1.8.5"},
        {syntax_tools, "1.6.7"},

        % Other libs
        {erlsom, "1.2.1"},
        {mochiweb, "0.167.10"},
        {nitrogen, "2.0.20100531.14"},
        {nprocreg, "0.1"},
        {simple_bridge, "1.0.2"},

        % Semiocast
        {analyzer, "22", load},
        {alien_models, "50", load},
        {alien_uniform_streams, "7", load},
        {api_server, "109", load},
        {aspell, "4", load},
        {binlog, "26", load},
        {certificate_authority, "8", load},
        {chasen, "11", load},
        {chinese_segmenter, "1", load},
        {commonlib, "250", permanent},  % always start commonlib.
        {ctl, "29", permanent},         % always start ctl.
        {dashboard_engine, "55", load},
        {dashboard_storage, "56", load},
        {dashboard_website, "226", load},
        {developer_website, "36", load},
        {engine, "455", load},
        {gate, "79", load},
        {geodb, "5", load},
        {geoip, "2", load},
        {image_magick, "8", load},
        {kdtree, "3", load},
        {kqueue, "1", load},
        {link_grammar_parser, "12", load},
        {memcached, "5", load},
        {mysql, "8", load},
        {nagios, "6", permanent},       % always start nagios.
        {opennlp, "8", load},
        {pgsql, "2", load},
        {pubsubhubbub, "4", load},
        {qr_website, "1", load},
        {re2, "1", load},
        {s_http, "42", load},
        {setproctitle, "2", permanent}, % always start setproctitle.
        {sink, "15", load},
        {sqlite, "9", load},
        {storage, "226", load},
        {svg, "12", load},
        {svm, "1", load},
        {text_ident, "27", load},
        {text_proc, "62", load},
        {titema_website, "7", load},
        {url_server, "8", load},
        {uuid, "6", load},
        {web_common, "14", load},
        {web_gate, "8", load},
        {web_storage, "5", load},
        {wikimedia, "6", load}
    ]}.
```

A few things we wish we had known about Erlang

Mistake #1:

Creating an erlang process to do a lot of work

- processes should spend most of their time waiting for messages (`gen_server`), or do some intensive work and quickly exit when done (`spawn_link`)
- when required, benchmark, as message passing with the worker process can prove expensive

```
Self = self(),
spawn_link(fun() -> Self ! {language, analyze_language(Text, MD0, Mode)} end),
spawn_link(fun() -> Self ! {location, analyze_location(MD0, Mode)} end),
{NProcessedLang, Language} = receive {language, RespLa}  -> RespLa  end,
{NProcessedLoc,  Location} = receive {location, RespLoc} -> RespLoc end,
```

Faster

```
{NProcessedLang, Language} = analyze_language(Text, MD0, Mode),
{NProcessedLoc,  Location} = analyze_location(MD0, Mode),
```

Mistake #2:

Creating a lot of processes for parallelized computing

- having more worker processes than schedulers does not make sense
- it can actually hurt, because processes waiting for a reply may not have it in time and will fail with a timeout

Thinking OTP

Mistake #3:

Starting processes outside the supervision tree

- `gen_server` & co. should be used everywhere, except for very short lived processes (that won't be upgraded)
- Every `gen_server` should be started from a supervisor
- A real-world supervision design requires `process_flag(trap_exit, true)`, `monitor/2` and `link/1`, as well as some thinking

Mistake #4:

Thinking obscure comments in the documentation do not really apply

- When in doubt, source code is handy, and helps figuring out when we really need to go off the rule

```
As a rule of thumb Modules should be a list with
one element [Module], where Module is the callback
module, if the child process is a supervisor,
gen_server or gen_fsm
```

erl -man supervisor

gen_manager is a gen_server with a callback module handling code_change messages (here, user_manager)

```
UserManagerSpec = {user_manager, % id
    {user_manager, start_link, []}, % init function
    transient, % restart children that crash
    ?SHUTDOWN_DELAY, worker,
    [gen_manager, user_manager] % modules
    },
```

Mistake #5:

Putting everything in a single virtual machine (node) per server

- Virtual machines may crash
- Code changes can fail and take down the whole node
- It's better to separate critical code
  - even per server if a crashing node can take a huge amount of RAM and make other nodes swap

Interfacing with foreign code

Foreign code

Six ways to interface foreign code with Erlang:

- Linked-In drivers
- External drivers
- NIFs
- os:cmd/1
- C-based distributed node
- Java-based distributed node (jinterface)

We tried them all…

…and are looking forward to future extensions to the native interface (R15?)

Foreign code

Method	We use/used for
Linked-in drivers	- aspell - kqueue (FreeBSD/MacOS X kqueue binding) - SQLite
External drivers	- ImageMagick - GeoIP - WebKit
NIFs	- uuid - re2 (linear time bound replacement for re) - bzip2
os:cmd/1	- OpenSSL - Batik (svg rasterizer in Java)
C-based distributed node	- ruby (we actually bound Rails websites with Erlang at some point)
jinterface	- OpenNLP

Foreign code

Mistake #6:

Using linked-in drivers for open source code that could crash/abort

E.g.: ImageMagick will abort on bad input

- External drivers are more suitable when external library is large, crash-prone or could leak (sometimes, the leak is in the glue…)
- Passing pointers is possible but requires some logic, typically binding a pointer to the port

Foreign code

Mistake #7:

Using linked-in drivers for I/O intensive code

E.g.: sqlite

- Linked-in driver code is executed within a scheduler thread. Running for too long will starve other processes that will timeout, waiting for messages
- Theoretically, we can use async threads (and we do with sqlite). However, enabling async threads (+A) has a huge impact on built-in I/O drivers
- Performance could be worse with external drivers

Erlang technologies we love

dialyzer

- part of our compilation cycle
- found many bugs, typically inconsistencies between callers and callees
- starting with `-spec` when defining exported funs

We wish it would be fixed/improved:

- horribly slow
- sometimes blind
- hard to understand
- fails on code_change code
- useless warnings that cannot be disabled

Erlang technologies we ~~love~~ hate

snmp

- makes it really easy to integrate erlang nodes within a monitoring solution (we use nagios and munin)

HiPE

OTP installed with --enable-native-lib and all our code compiled with +native

- helps with CPU-bound work (including dialyzer…)
- most patches we submitted were HiPE-related

We are also grateful to the authors of:
erlsom, mochiweb, nitrogen, zotonic…

Thank you !

paul@semiocast.com