facebook

Functional Programming at Facebook

Chris Piro, Eugene Letuchy
Commercial Users of Functional Programming (CUFP)
Edinburgh, Scotland
4 September 2009

Agenda

1. Facebook and Chat
2. Chat architecture
3. Erlang strengths
4. Setbacks
5. What has worked

Facebook

The Facebook Environment

The Facebook Environment

- The web site
  - More than 250 million active users
  - More than 3.5 billion minutes are spent on Facebook each day

The Facebook Environment

- The web site
  - More than 250 million active users
  - More than 3.5 billion minutes are spent on Facebook each day

- The engineering team
  - Fast iteration: code gets out to production within a week
  - Polyglot programming: interoperability with Thrift
  - Practical: high-leverage tools win

Using FP at Facebook

Using FP at Facebook

- Erlang
  - Chat backend (channel servers)
  - Chat Jabber interface (ejabberd)
  - AIM presence: a JSONP validator

Using FP at Facebook

- Erlang
  - Chat backend (channel servers)
  - Chat Jabber interface (ejabberd)
  - AIM presence: a JSONP validator

- Haskell
  - lex-pass: PHP parse transforms
  - Lambdabot
  - textbook: command line Facebook API client
  - Thrift binding

Thrift

Thrift

- An efficient, cross-language serialization and RPC framework

Thrift

- An efficient, cross-language serialization and RPC framework
- Write interoperable servers and clients

Thrift

- An efficient, cross-language serialization and RPC framework
- Write interoperable servers and clients
- Includes library and code generator for each language

Thrift

- An efficient, cross-language serialization and RPC framework
- Write interoperable servers and clients
- Includes library and code generator for each language
- Servers define interfaces with an IDL

```
struct UserProfile {
  1: i32 uid,
  2: string name,
  3: string blurb
}

service UserStorage {
  void store(1: UserProfile user),
  UserProfile retrieve(1: i32 uid)
}
```

Thrift

- An efficient, cross-language serialization and RPC framework
- Write interoperable servers and clients
- Includes library and code generator for each language
- Servers define interfaces with an IDL
- Many supported languages

```
struct UserProfile {
  1: i32 uid,
  2: string name,
  3: string blurb
}

service UserStorage {
  void store(1: UserProfile user),
  UserProfile retrieve(1: i32 uid)
}
```

C++	Objective C	Ruby
C#	OCaml	Squeakr
Erlang	Perl	
Haskell	PHP	HTML
Java	Python	XSD

Facebook Chat

Motivation

Sasha Rush wrote
at 10:47pm on May 31st, 2008

deleted upon pictureds request. still think we should keep the name.

Write on Sasha's Wall

Daniel Corson wrote
at 10:40pm on May 31st, 2008

did you use the ConeyTypeInference app to make that

Write on Daniel's Wall

Daniel Corson wrote
at 10:02pm on May 31st, 2008

the JuliaLambdas

Write on Daniel's Wall

Sasha Rush wrote
at 10:47pm on May 30th, 2008

Also we need a nerdy, groan inducing name. I'm thinking ZuckerLambda.

Write on Sasha's Wall

Sasha Rush wrote
at 10:45pm on May 30th, 2008

I was just going to send you that link. Let's do it. Although maybe we should try one of the old contests first. My practical haskell is still kind of slow.

Write on Sasha's Wall

Daniel Corson shared a link
at 9:38pm on May 30th, 2008

http://www.icfpcontest.org/

Motivation

Why does Facebook need Chat?

- Inbox, Wall, Comments are asynchronous, slow
- Real-time conversation
- Unique advantages:
  - List of friends for free
  - Integrated Facebook content
  - No install required

Timeline

Timeline

- Jan 2007: Chat prototyped at Hackathon

Timeline

- Jan 2007: Chat prototyped at Hackathon
- Fall 2007: Chat becomes a “real” project
  - 4 engineers, 0.5 designer

Timeline

- Jan 2007: Chat prototyped at Hackathon
- Fall 2007: Chat becomes a “real” project
  - 4 engineers, 0.5 designer
- Winter 2007-08: Code, code, code (learn Erlang)

Timeline

- Jan 2007: Chat prototyped at Hackathon
- Fall 2007: Chat becomes a “real” project
  - 4 engineers, 0.5 designer
- Winter 2007-08: Code, code, code (learn Erlang)
- Feb 2008: “Dark launch” testing begins
  - Simulates load on the Erlang servers … they hold up

Timeline

- Jan 2007: Chat prototyped at Hackathon
- Fall 2007: Chat becomes a “real” project
  - 4 engineers, 0.5 designer
- Winter 2007-08: Code, code, code (learn Erlang)
- Feb 2008: “Dark launch” testing begins
  - Simulates load on the Erlang servers … they hold up
- Apr 6, 2008: First user message sent:

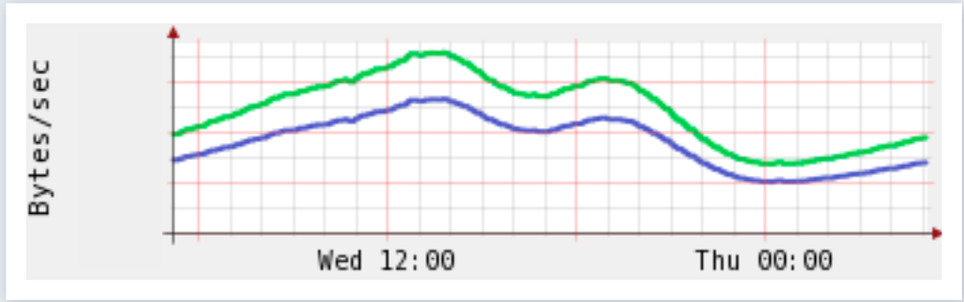
Timeline

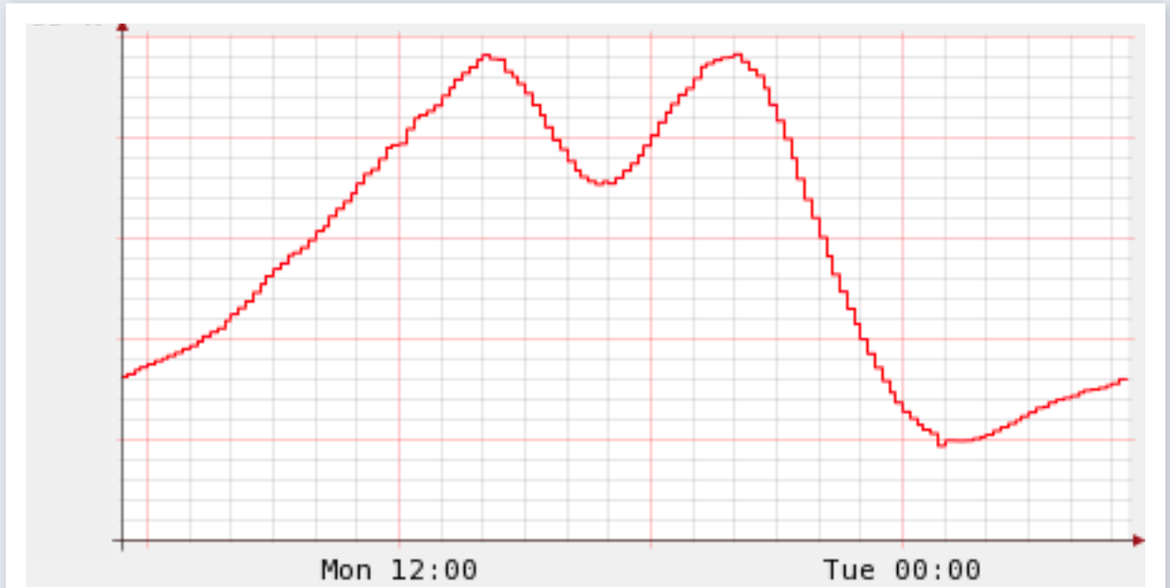
- Jan 2007: Chat prototyped at Hackathon
- Fall 2007: Chat becomes a “real” project
  - 4 engineers, 0.5 designer
- Winter 2007-08: Code, code, code (learn Erlang)
- Feb 2008: “Dark launch” testing begins
  - Simulates load on the Erlang servers … they hold up
- Apr 6, 2008: First user message sent: “msn chat?”

Timeline

- Jan 2007: Chat prototyped at Hackathon
- Fall 2007: Chat becomes a “real” project
  - 4 engineers, 0.5 designer
- Winter 2007-08: Code, code, code (learn Erlang)
- Feb 2008: “Dark launch” testing begins
  - Simulates load on the Erlang servers … they hold up
- Apr 6, 2008: First user message sent: “msn chat?”
- Apr 23, 2008: 100% rollout (Facebook has 70M users at the time)

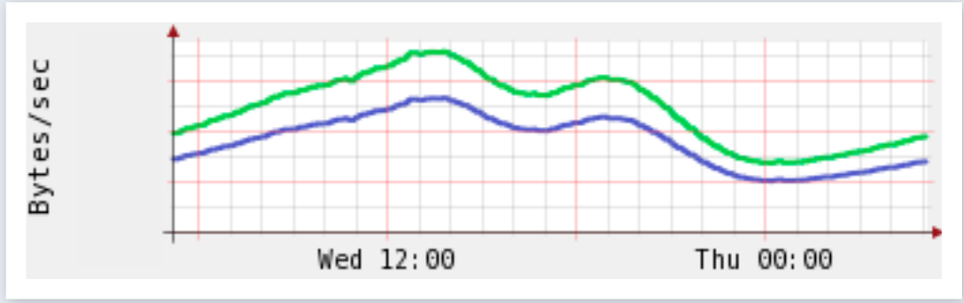
Chat today

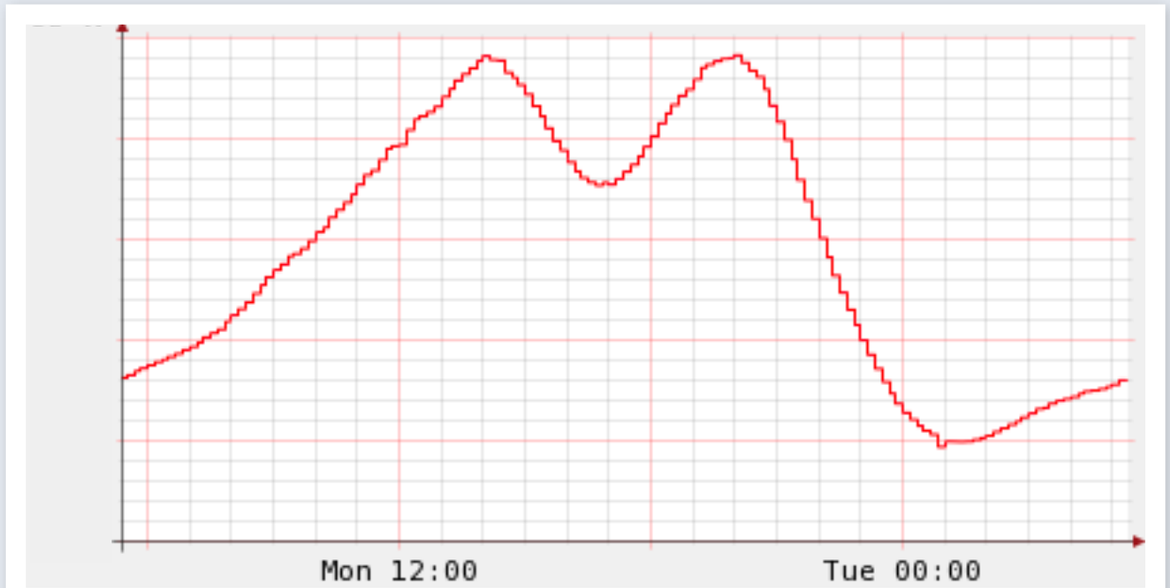




Chat today

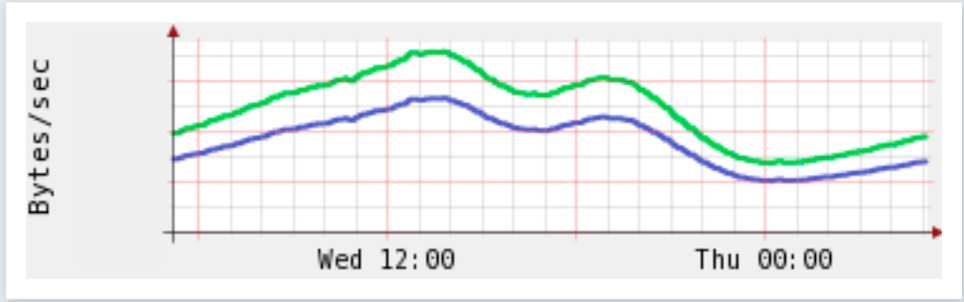
- 1+ billion user messages / day
- 10+ million active channels at peak
- 1+ GB traffic at peak
- 100+ channel machines

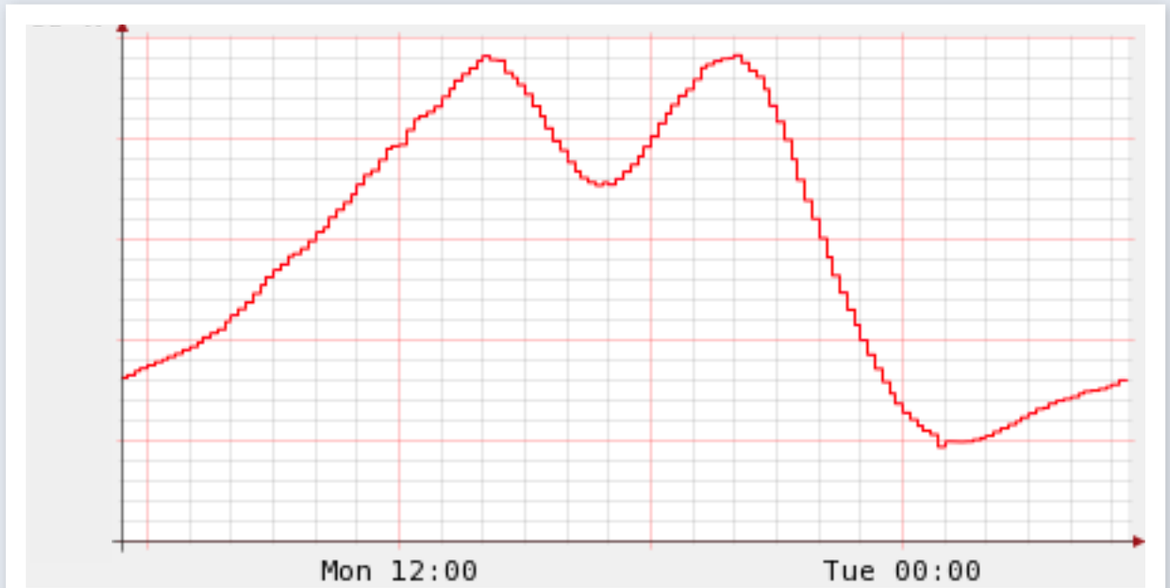




Chat today

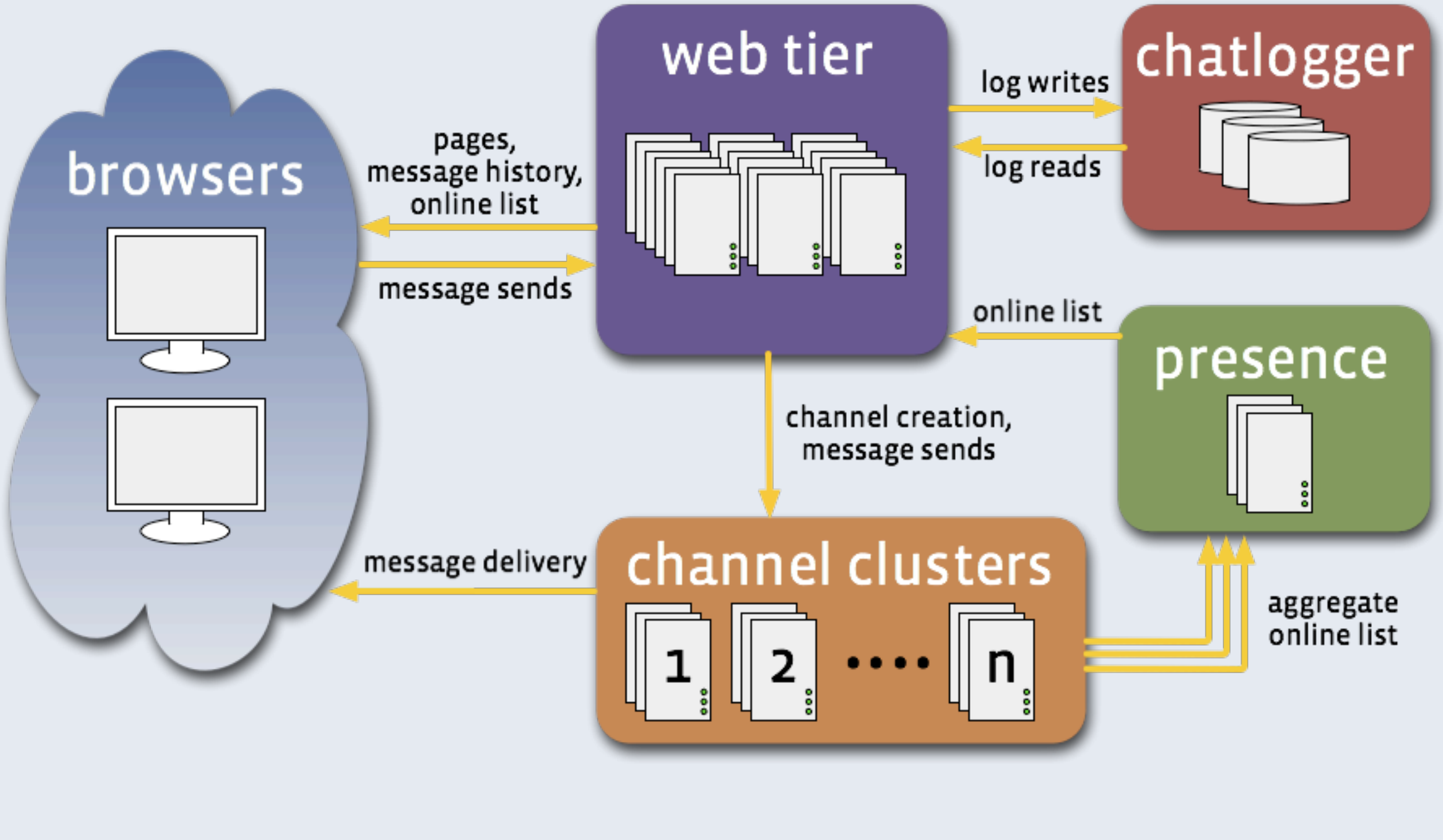
- 1+ billion user messages / day
- 10+ million active channels at peak
- 1+ GB traffic at peak
- 100+ channel machines
- Work load has increased 10x while machines not even 3x



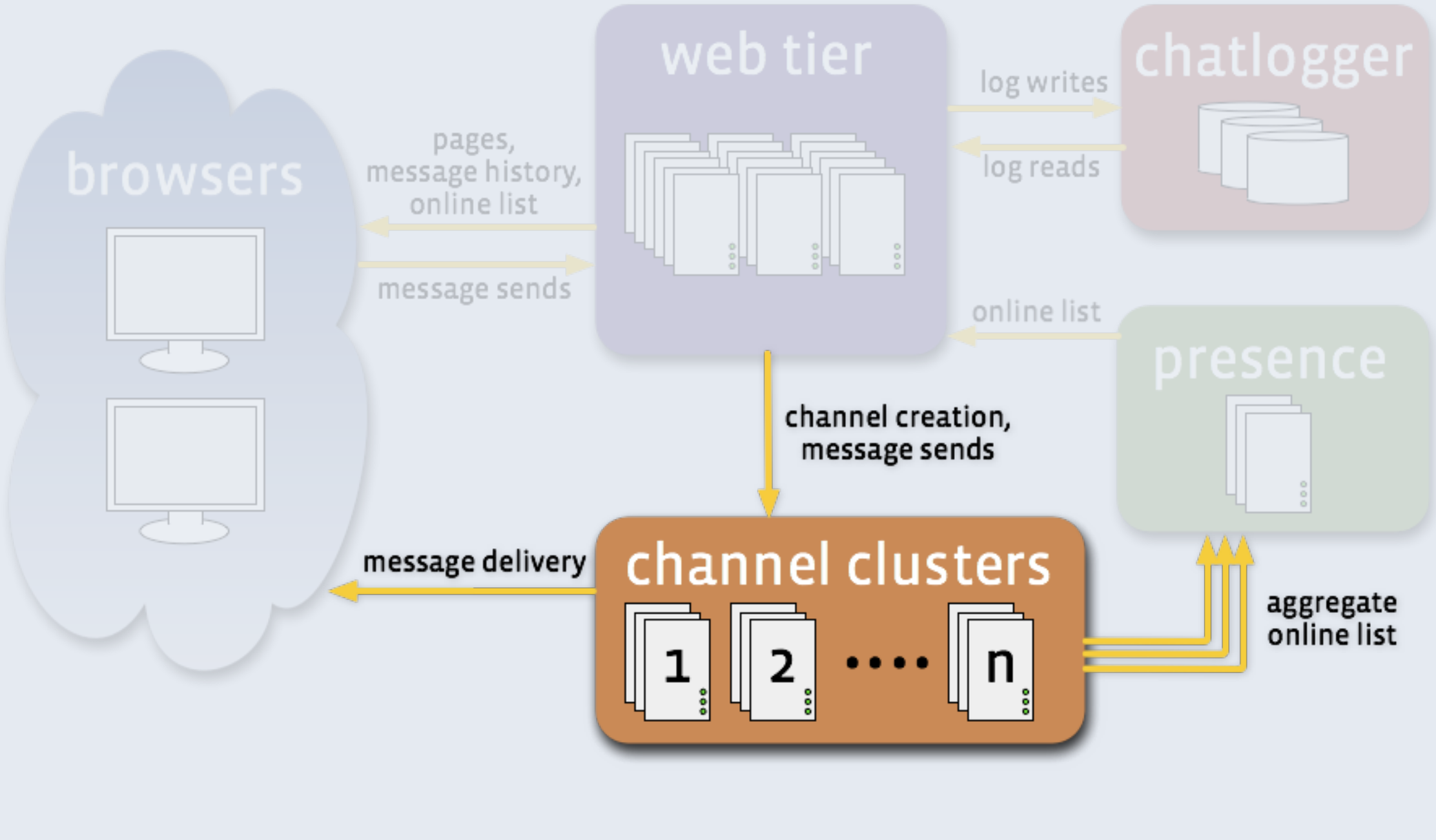


Chat Architecture

Channel servers (Erlang)



Channel servers (Erlang)



Channel servers (Erlang)

Channel servers (Erlang)

Architectural overview

- Web, Jabber tiers authenticate, deliver messages

Channel servers (Erlang)

Architectural overview

- Web, Jabber tiers authenticate, deliver messages
- One message queue per user (channel)

Channel servers (Erlang)

Architectural overview

- Web, Jabber tiers authenticate, deliver messages
- One message queue per user (channel)
- Timing, idleness information

Channel servers (Erlang)

Architectural overview

- Web, Jabber tiers authenticate, deliver messages
- One message queue per user (channel)
- Timing, idleness information
- HTTP long poll to simulate push (Comet)
  - Server replies when a message is ready
  - One active request per browser tab

Channel servers (Erlang)

Architectural overview

- Web, Jabber tiers authenticate, deliver messages
- One message queue per user (channel)
- Timing, idleness information
- HTTP long poll to simulate push (Comet)
  - Server replies when a message is ready
  - One active request per browser tab
- User ID space partitioned statically (division of labor)

Channel servers (Erlang)

Architectural overview

- Web, Jabber tiers authenticate, deliver messages
- One message queue per user (channel)
- Timing, idleness information
- HTTP long poll to simulate push (Comet)
  - Server replies when a message is ready
  - One active request per browser tab
- User ID space partitioned statically (division of labor)
- Each partition served by a cluster of machines (availability)

Erlang strengths

Concurrency

Concurrency

- Cheap parallelism at massive scale

Concurrency

- Cheap parallelism at massive scale
- Simplifies modeling concurrent interactions
  - Chat users are independent and concurrent
  - Mapping onto traditional OS threads is unnatural

Concurrency

- Cheap parallelism at massive scale
- Simplifies modeling concurrent interactions
  - Chat users are independent and concurrent
  - Mapping onto traditional OS threads is unnatural
- Locality of reference

Concurrency

- Cheap parallelism at massive scale
- Simplifies modeling concurrent interactions
  - Chat users are independent and concurrent
  - Mapping onto traditional OS threads is unnatural
- Locality of reference
- Bonus: carries over to non-Erlang concurrent programming

Distribution

Distribution

- Connected network of nodes

Distribution

- Connected network of nodes
- Remote processes look like local processes
  - Any node in a channel server cluster can route requests
  - Naive load balancing

Distribution

- Connected network of nodes
- Remote processes look like local processes
  - Any node in a channel server cluster can route requests
  - Naive load balancing
- Distributed Erlang works out-of-the-box (all nodes are trusted)

Fault Isolation

Fault Isolation

- Bugs in the initial versions of Chat:
  - Process leaks in the Thrift bindings
  - Unintended multicasting of messages
  - Bad return state for presence aggregators

Fault Isolation

- Bugs in the initial versions of Chat:
  - Process leaks in the Thrift bindings
  - Unintended multicasting of messages
  - Bad return state for presence aggregators
- (Horrible) bugs don't kill a mostly functional system:
  - C/C++ segfault takes down the OS process and your server state
  - Erlang badmatch takes down an Erlang process
    - … and notifies linked processes

Error logging (crash reports)

Error logging (crash reports)

- Any proc_lib-compliant process generates crash reports

Error logging (crash reports)

- Any proc_lib-compliant process generates crash reports
- Error reports can be handled out-of-band (not where generated)

Error logging (crash reports)

- Any proc_lib-compliant process generates crash reports
- Error reports can be handled out-of-band (not where generated)
- Stacktraces point the way to bugs (functional languages win big here)

Error logging (crash reports)

- Any proc_lib-compliant process generates crash reports
- Error reports can be handled out-of-band (not where generated)
- Stacktraces point the way to bugs (functional languages win big here)
- Writing error_log handlers is simple:
  - gen_event behavior
  - Allows for massaging of the crash and error messages (binaries!)
  - Thrift client in the error log

Error logging (crash reports)

- Any proc_lib-compliant process generates crash reports
- Error reports can be handled out-of-band (not where generated)
- Stacktraces point the way to bugs (functional languages win big here)
- Writing error_log handlers is simple:
  - gen_event behavior
  - Allows for massaging of the crash and error messages (binaries!)
  - Thrift client in the error log

- WARNING: excessive error logging can OOM the Erlang node!

Hot code swapping

Hot code swapping

- Restart-free upgrades are awesome (!)
  - Pushing new functional code for Chat takes ~20 seconds
  - No state is lost

Hot code swapping

- Restart-free upgrades are awesome (!)
  - Pushing new functional code for Chat takes ~20 seconds
  - No state is lost
- Test on a running system

Hot code swapping

- Restart-free upgrades are awesome (!)
  - Pushing new functional code for Chat takes ~20 seconds
  - No state is lost
- Test on a running system
- Provides a safety net … rolling back bad code is easy

Hot code swapping

- Restart-free upgrades are awesome (!)
  - Pushing new functional code for Chat takes ~20 seconds
  - No state is lost
- Test on a running system
- Provides a safety net … rolling back bad code is easy

- NOTE: we don’t use the OTP release/upgrade strategies

Monitoring and Error Recovery

Monitoring and Error Recovery

- Supervision hierarchies
  - Organize (and control) processes
  - Systematize restarts and error recovery
  - Extended supervisor with a “directory” type
    - one_for_one with string -> child pid map

Monitoring and Error Recovery

- Supervision hierarchies
  - Organize (and control) processes
  - Systematize restarts and error recovery
  - Extended supervisor with a “directory” type
    - one_for_one with string -> child pid map
- net_kernel (Distributed Erlang)
  - sends nodedown, nodeup messages
  - any process can subscribe

Monitoring and Error Recovery

- Supervision hierarchies
  - Organize (and control) processes
  - Systematize restarts and error recovery
  - Extended supervisor with a “directory” type
    - one_for_one with string -> child pid map
- net_kernel (Distributed Erlang)
  - sends nodedown, nodeup messages
  - any process can subscribe
- heart: monitors and restarts the OS process

Hibernation

Hibernation

- Drastically shrink memory usage with erlang:hibernate/3
  - Throws away the call stack, minimizes heap
  - Enters a wait state for new messages
  - “Jumps” into a passed-in function for a received message

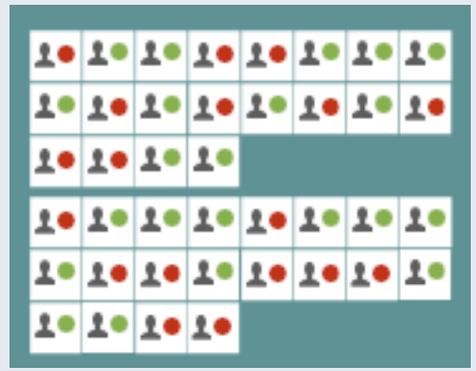
Hibernation

- Drastically shrink memory usage with erlang:hibernate/3
  - Throws away the call stack, minimizes heap
  - Enters a wait state for new messages
  - "Jumps" into a passed-in function for a received message
- Perfect for a long-running, idling HTTP request handler

Hibernation

- Drastically shrink memory usage with erlang:hibernate/3
  - Throws away the call stack, minimizes heap
  - Enters a wait state for new messages
  - “Jumps” into a passed-in function for a received message
- Perfect for a long-running, idling HTTP request handler
- But … not compatible with gen_server:call (and gen_server:reply)
  - gen_server:call has its own receive() loop
  - hibernate() doesn’t support an explicit timeout
  - gen_hibernate: a few hours and a look at gen.erl

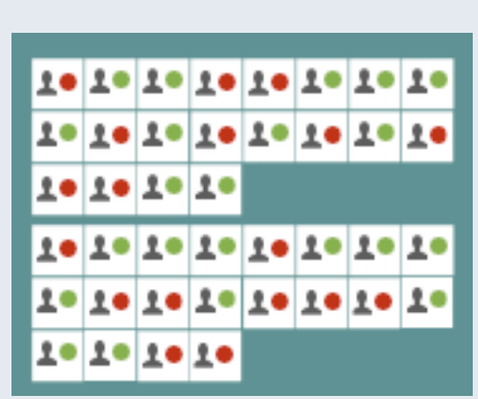
hipe_bifs



hipe_bifs

Cheating single assignment

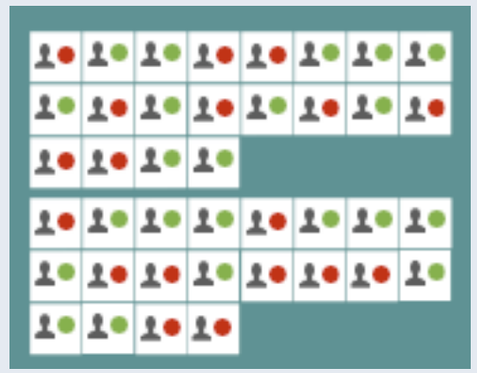
- Erlang is opinionated:
  - Destructive assignment is hard because it should be



hipe_bifs

Cheating single assignment

- Erlang is opinionated:
  - Destructive assignment is hard because it should be
- hipe_bifs:bytearray_*(): manipulate references to mutable arrays (!)
  - Necessary for aggregating Chat users' presence
  - Same in-memory format as presence servers (C++)
  - Don't tell anyone!



Setbacks

“What’s Erlang?”

“What’s Erlang?”

- Lack of Erlang educational resources (at start of 2007)
  - Few industry-focused English-language resources
  - Few blogs (outside of Yariv’s and Joel Reymont’s)
  - U.S. Erlang community limited in number and visibility

“What’s Erlang?”

- Lack of Erlang educational resources (at start of 2007)
  - Few industry-focused English-language resources
  - Few blogs (outside of Yariv’s and Joel Reymont’s)
  - U.S. Erlang community limited in number and visibility
- Engineers are uncomfortable with FP
  - Universities have very conservative curricula
  - FP : academia, AI :: ‘normal programming’ : industry
  - “If you want to succeed, learn C++ and Java”, not “use the right tool for the job”
  - Not similar to rest of the codebase, not hiring specifically for FP

Institutional pressures

Institutional pressures

- Hard to get others to join the effort

Institutional pressures

- Hard to get others to join the effort
- Can’t reuse specialized infrastructure
  - PHP-, C++-centric tools
  - Chat deploy process is a one-off

Institutional pressures

- Hard to get others to join the effort
- Can’t reuse specialized infrastructure
  - PHP-, C++-centric tools
  - Chat deploy process is a one-off
- Divides department into us vs. them
  - We’re “the Erlang guys”
  - Sole responsibility for fixing bugs
  - Less time for us to evangelize and innovate elsewhere

Institutional pressures

- Hard to get others to join the effort
- Can’t reuse specialized infrastructure
  - PHP-, C++-centric tools
  - Chat deploy process is a one-off
- Divides department into us vs. them
  - We’re “the Erlang guys”
  - Sole responsibility for fixing bugs
  - Less time for us to evangelize and innovate elsewhere
- (Seemingly) contrary to “move fast” value

What has worked

What has worked

What has worked

- Use FP from the beginning

What has worked

- Use FP from the beginning
- Outline language strengths, give evidence

What has worked

- Use FP from the beginning
- Outline language strengths, give evidence
- Internal tech talks

What has worked

- Use FP from the beginning
- Outline language strengths, give evidence
- Internal tech talks
- ICFP Programming Contest: give the FP people an excuse!

What has worked

- Use FP from the beginning
- Outline language strengths, give evidence
- Internal tech talks
- ICFP Programming Contest: give the FP people an excuse!
- Language independence with Thrift

What has worked

- Use FP from the beginning
- Outline language strengths, give evidence
- Internal tech talks
- ICFP Programming Contest: give the FP people an excuse!
- Language independence with Thrift
- “The right tool for the job”

facebook