

An Erlang-Based Philosophy for Service Reliability

Jamshid Mahdavi

March 10, 2016

About me

Networking, TCP, Proxies

@WhatsApp since 2013

First experience with Erlang

Work on media servers and media delivery

About WhatsApp

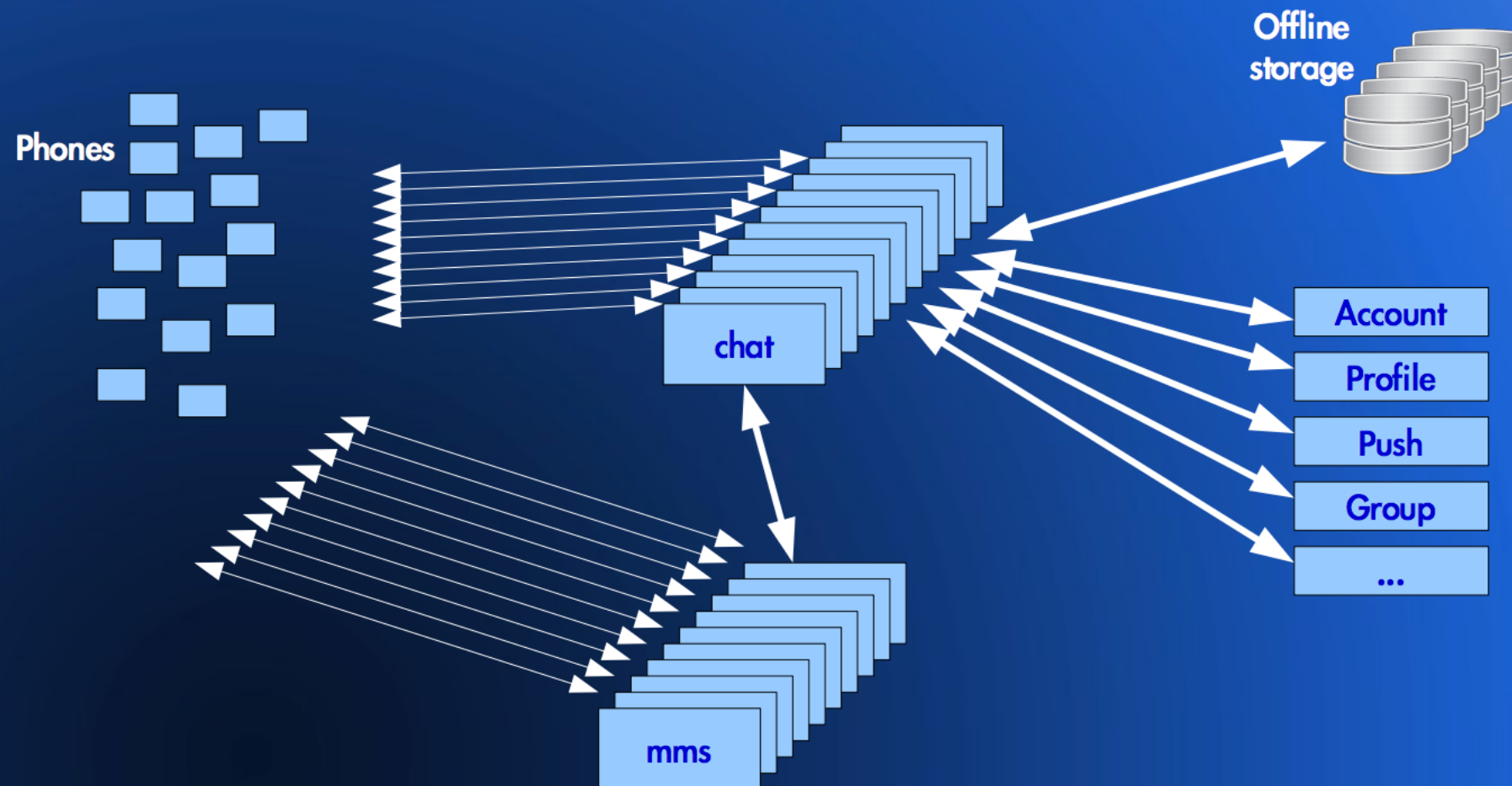
Messaging app including chat, media delivery, voice calling

Bought by Facebook in 2014

Small team

Growing user base

System Overview



Specs

Bare metal servers

Mostly dual CPU, 128GB – 768GB

Quad NIC

FreeBSD 10

Erlang R16B

Why This Talk?

Not really a technical talk

From our contacts at facebook
we are “known for reliability”

Topics

- Development Practices
- Deployment Practices
- Handling Failures
- Monitoring and Alerts

Reliability: Setting Expectations

Rule #1: Never lose a message

Rule #2: Availability trumps everything

Six nines rule: Fail < 1 in a million

Development

Minimal footprint

Use only the software we absolutely need

```
$ pstree
-+= 00001 root /sbin/init --
  |--= 00856 root /sbin/devd
  |--= 01049 root /usr/sbin/syslogd -ss
  |--= 01197 root /usr/sbin/ntpd -c /etc/ntp.conf -p /var/run/ntpd.pid -f /var/db/
ntpd.drift
  -+= 01219 root /usr/bin/perl -w /usr/local/bin/wsar -H -log (perl5.16.3)
    \--- 01220 root /usr/bin/perl -w /usr/local/bin/wsar -H -log (perl5.16.3)
  -+= 01234 root /usr/sbin/cron -m -s
    \-+- 74065 root cron: running job (cron)
      \---= 74066 root /usr/bin/perl -w /usr/local/bin/wsar -all -log (perl5.16.3)
  --- 17433 whatsapp /usr/local/lib/erlang/erts-5.10.1/bin/epmd -daemon
  +- 17435 whatsapp /usr/local/lib/erlang/erts-5.10.1/bin/beam.smp -zdbbl 16384 -
swt very_low -sbt tnnps -sbwt none -P 7200000 -- -root /usr/local/lib/erlang -
progname erl -- -home
  \-+= 17447 whatsapp inet_gethost 4
    --- 46176 whatsapp inet_gethost 4
    --- 58420 whatsapp inet_gethost 4
    --- 58422 whatsapp inet_gethost 4
    \--- 58423 whatsapp inet_gethost 4
  --= 18544 whatsapp panam/bin/panam
  --= 01299 root /usr/libexec/getty 3wire.19200 ttyu1
  --= 01291 root /usr/libexec/getty Pc ttyv0
    [+ 7 more]
```

Development

Minimal footprint

Use only the software we absolutely need

“Be careful”

I.e.: don't make mistakes

This is intended to capture a lot of things that we *don't* do
test automation, code reviews, etc.

Development

Minimal footprint

Use only the software we absolutely need

“Be careful”

I.e.: don't make mistakes

This is intended to capture a lot of things that we *don't* do
test automation, code reviews, etc.

Investigate every bug

Benefits of Erlang

Compact code base (20k LOC for media servers)

Silo architecture

Failures isolate to one feature – minimize user-visible impact

Server migrations

Good opportunities for rewriting / refactoring systems

Deployment Practices

Automation

- View as a way to minimize human effort

- NOT trying to take humans out of the loop (in most cases)

All deploys are manual

- Friction by design

- Erlang hot load – zero dropped connections or requests

Lots of small / simple deploys

- Change slowly

- Value stability

Failure Modes

Hardware

Ram, disks, nics are common
Occasional more esoteric stuff

Network

Work with our vendor on keeping this perfect

Software

Bugs do happen
Problem scope is usually apparent

Monitoring

One monitoring script (“mon.sh”) runs on all of our servers

Hardware and certain classes of software issues (e.g. backlog)
1005 LOC; + extra 600 LOC for “disk” systems

Several external health check scripts

mms_mon, www_mon, dns_mon, cdn_mon, ...

Special full mesh monitoring for networking

Used for debugging backend network problems

Trend-based alerts

Percent errors, total traffic, etc.

Handling Alerts

Alerts are “broadcast” on WhatsApp to whole team

- Lucky people also get SMS

- Ring every minute until fixed

“Fix Fast” vs. “Deep Redundancy”

- Expect to fix problems when they happen

Don't try to build most systems to be resilient to double faults

- Since 2010, no completely lost partitions

- Occasionally have temporary partition unavailable or inconsistent

Add new alarms whenever we find an issue which doesn't alert us

Summary

Keep it small, keep it simple

Complexity makes systems harder to maintain and debug

#letitcrash

Don't spend a lot of effort handling things which aren't supposed to happen

Zero bugs / Fix Fast

The longer a bug lingers, the more it costs

Questions?



WhatsApp