Let us know what you think

Click ‘engage’ to rate sessions and ask questions

Follow us on Twitter @GOTOber}

www.gotober.com
Thinking in a Highly Concurrent, Mostly-functional Language

GOTO Berlin
Berlin, December 4th 2015

Francesco Cesarini
Founder & Technical Director
@francescoC
francesco@erlang-solutions.com
Thinking in a Highly Concurrent, Mostly-functional Language

QCON London, March 12th, 2009

Francesco Cesarini
francesco@erlang-consulting.com
counter_loop(Count) ->
        receive
            increment ->
                counter_loop(Count + 1);
            {count, To} ->
                To ! {count, Count},
                counter_loop(Count)
        end.
After you’ve opened the top of your head, reached in and turned your brain inside out, this starts to look like a natural way to count integers. And Erlang does require some fairly serious mental readjustment.

However... having spent some time playing with this, I tell you...

Tim Bray, Director of Web Technologies - Sun Microsystems
... If somebody came to me and wanted to pay me a lot of money to build a large scale message handling system that really had to be up all the time, could never afford to go down for years at the time, I would unhesitatingly choose Erlang to build it in.

Tim Bray, Director of Web Technologies - Sun Microsystems
Syntax
Concurrency
Two ways to do concurrency

Mutable State

Immutable State
Two ways to do concurrency

Problem 1 with mutable state:
Your program crashes whilst executing in the critical section…
Two ways to do concurrency

Problem 2 with mutable state:
Where do you locate your state…

Seattle  London
Two ways to do concurrency

**Problem 3 with mutable state:**
What happens if your network connectivity fails...

- Seattle
- London
Two ways to do concurrency

Problem 1 with mutable state:

Your program crashes whilst executing in the critical section…

Your state does not get corrupted.
Two ways to do concurrency

Problem 2 with mutable state:
Where do you locate your state…

You do not Locate state, you copy it.
Two ways to do concurrency

Problem 3 with mutable state:
What happens if your network connectivity fails...

Make sure your business logic and databases handle network splits!
Creating a new process using spawn

-module(ex3).
-export([activity/3]).

activity(Name,Pos,Size) ->
...........

Pid = spawn(ex3,activity,[Joe,75,1024])
Erlang Highlights: Concurrency

Processes communicate by asynchronous message passing

Pid ! {data,12,13}

receive
  {start} -> ........
  {stop} -> ........
  {data,X,Y} -> ........
end
Products: AXD301 Switch - 1996

A Telephony-Class, scalable (10 - 160 GBps) ATM switch

Designed from scratch in less than 3 years

AXD 301 Success factors:
- Competent organisation and people
- Efficient process
- Excellent technology (e.g. Erlang/OTP)
Erlang: ca 1.5 million lines of code
- Nearly all the complex control logic
- Operation & Maintenance
- Web server and runtime HTML/JavaScript generation

C/C++: ca 500k lines of code
- Third party software
- Low-level protocol drivers
- Device drivers

Java: ca 13k lines of code
- Operator GUI applets
Concurrency Modeling

Model for the natural concurrency in your problem

In the old days, processes were a critical resource
- Rationing processes led to complex and unmanageable code

Nowadays, processes are very cheap: if you need a process - create one!

Example: AXD301 process model

1\textsuperscript{st} prototype:
- 6 processes/call
- 2 processes/call
- 1 process/all calls
- 2 processes/call transaction
- 4-5 processes/call transaction
1+1 Redundancy - Good ol’ Telecoms

~ 35 000 calls per processor pair

No ongoing sessions lost at “failover”

Stable-state replication

Active

Standby

Device board

Control plane

User plane

Control signalling

Data path
First IM Proxy Prototype - 2000

sockets

multiplexing

state/error handling

de-multiplexing

sockets

users

jabber

jabber

de-multiplexing

multiplexing

listener
First IM Proxy Prototype - 2000

- multiplexing
- de-multiplexing
- state/error handling
- sockets
- users

supervisor

simple 1-1

listener
Products: EjabberD IM Server - 2002

A distributed XMPP server

Started as an Open Source Project by Alexey Shchepin

Commercially Supported by Process-One (Paris)
- 40% of the XMPP IM market
- Used as a transport layer
- Managed 30,000 users / node
Products: EjabberD IM Server - 2002

A distributed XMPP server

Started as an Open Source project by Alexey Shchepin

Commercially Supported by Process-One (Paris)
- 40% of the XMPP IM market
- Used as a transport layer
- 2008, Managed 30,000 users / node

MongooseIM is a fork and rewrite
- Open Source, supported by Erlang Solutions
- Used for Messaging and Device Management
- 2014, managed 1 million users / node
Fully Replicated Cluster - Ejabberd 2002

Client must re-connect if one of its session handlers dies

Fully replicated Mnesia database
MMGS - Messaging Gateway - 2008
MMGS- Messaging Gateway - 2008
Erlang Concurrency Under Stress - Pre-SMP

Throughput / Second

Simultaneous Requests

100% CPU

Line 1 Balanced Erlang System

Line 2 Erlang System with bottle necks
Erlang Concurrency Under Stress - Pre-SMP

YAWS Throughput
(KBytes/second)
Stress Tests With SMP

I/O Starvation
TCP/IP Congestion
Memory Spikes
Timeout Fine-tuning
OS Limitations
ERTS Configuration Flags
Shut down Audit Logs
SMP bottlenecks - pre 2008
SMP bottlenecks - post 2008

Erlang VM

Scheduler #1
run queue

Scheduler #2
run queue

...  

Scheduler #N
run queue

migration logic
Big Bang Benchmark - post 2008

Red: Single Queue, Blue: Multiple Run Queue on a Tilera TilePro64 (64 cores)
Mandelbrot throughput

- **Stack**
- **Concurix Aug 8**
- **Concurix June 25**

Cores

Cells/second

- 0
- 4,000,000
- 8,000,000
- 12,000,000
- 16,000,000
Now for the Bottlenecks

Build your next website with Erlang — the world’s most advanced networking platform.

Do you pine for a simpler time when web pages loaded in under one second? Chicago Boss is the answer to slow server software: a Rails-like framework for Erlang that delivers web pages to your users as quickly and efficiently as possible.
Now for the Bottlenecks
Now for the Bottlenecks

The gen_server can become a serialization bottleneck, particularly with `gen_server:call(...)`. 
Ahmdal’s Law

\[ T(n) = T(1) \left( B + \frac{1}{n} (1 - B) \right) \]

- \( n \) - the number of threads of execution
- \( B \) - the fraction of the algorithm that is strictly serial
- \( n \) - Number of parallel threads
- \( T(n) = \text{The time an algorithm to finish when being executed on } n \text{ thread(s)} \)
Now for the Bottlenecks

www.concurix.com
Now for the Bottlenecks

Tons of headroom still!

Chicago Boss scaling

- Unoptimized
- Caching
- Caching + Refactoring

Users/sec vs. Schedulers
Heterogeneous multi-core hardware is here to stay

Different cores doing different things
CPUs, GPUs, FPGA

Parallella Board
Dual core ARM processor + FPGA
1GB RAM + MicroSD Card

16 or 64 core Epiphany co-processor
Gigabit Ethernet
2x USB ports + HDMI port
Erlang now runs on 32KB Epiphany thanks to heroic efforts of Kostis and Magnus at Uppsala...P2=epiphany:spawn(..)
mlang.se/presentation.p...
The Fastest Computer in the World!

Tianhe-2
Chinese National University of Defence Technology

- 33.86 petaflops/s (November 2013)
- 16,000 Nodes, each with 2 Ivy Bridge multicore and 3 Xeon Phi
- 3,120,000 x86 cores in total
Riak and other scalable architectures

- A ring with 32 partitions
- Hash("artist", "REM")
- A single vnode/partition
N/R/W Values

```
put(<<"artist">>, <<"REM">>)
```

(N=3)
N/R/W Values

get/put("artist", "REM", R/W=2)

{ok, Object}

node 0
node 1
node 2
node 3
Clusters and SD Erlang

- **Two major issues**
  - Fully connected clusters
  - Explicit process placement
- **Scalable Distributed (SD) Erlang**
  - Nodes grouping
  - Non-transitive connections
  - Implicit process placement
  - Part of the standard Erlang/OTP package
- **New concepts introduced**
  - Locality, affinity and distance
“To scale the radical concurrency-oriented programming paradigm to build reliable general-purpose software, such as server-based systems, on massively parallel machines (10^5 cores).”
“Limitations exist on all levels. You would not want an Erlang VM to run with $10^5$ schedulers.”
Push the responsibility for scalability from the programmer to
the VM

Analyze performance and scalability

Identify bottlenecks and prioritize changes and extensions

Tackle well-known scalability issues

Ets tables (shared global data structure)

Message passing, copying and frequently communicating
processes
Thank You!

@francescoc
francesco@erlang-solutions.com
Thank You!

@francescoc
francesco@erlang-solutions.com

Discount Code: authd
50% off the Early Release
40% off the printed copy
Please Remember to rate session

Thank you!

Follow us on Twitter @GOTOber

www.gotober.com