Distributing Elixir apps

When one machine is not enough

Adrian Magdas

Co-Founder & Lead Developer @ Crafting Software
Innovation
adrian@craftingsoftware.com
@zenCrafter

Agenda

1. Actor model concurrency
2. Let it crash(again)
3. Distribution models

Elixir agents/processes

- Agents in Elixir are regular light weight processes
- are the key ingredient in building fault tolerant and scalable apps
- are shared nothing execution contexts which are PREEMTIVELY scheduled across a set of OS threads
- intrinsically capable of running concurrently
- In Elixir, all code runs inside processes. Processes are isolated from each other, run concurrent to one another and communicate via message passing

Anatomy of a process

- Message box
- Non shared working memory
- Code path & own execution flow

Message box

- process may receive arbitrary messages and store them in the process's inbox
- filtered using pattern matching, the process decides the order of consumption from message box
- message passing ties the small programs into bigger ones

Non shared memory(most of the time)

- each process has it's own block of RAM
- when a process is done, the memory if freed O(1)
- you have to be aware of this and think how to keep the state in short lived processes so you can reclaim the memory fast
- you can set memory limits per process

Process execution

- modules are the unit of code organization
- processes are the unit of code execution
- when the code stops or errors, the process exits

Message passing

- `send receiver, message`
- posts the message to the receiver's inbox and returns the message
- delivery is not guaranteed
- receiver can be a pid, an atom for a named process, a tuple {atom, node} or a port
- message can be anything

Lifespan tracking

- linking - connects one process to another, if the process exists abnormally, the linked process will also crash
- monitoring - handle exit events instead of crashing via `:trap_exit` process flag
- GenServer & Supervisor

GenServer

- a behavior module for implementing the server of a client-server relation.

- a GenServer is a process like any other Elixir process and it can be used to keep state, execute code asynchronously and so on. The advantage of using a generic server process (GenServer) implemented using this module is that it will have a standard set of interface functions and include functionality for tracing and error reporting. It will also fit into a supervision tree.

Supervisor

- Process that supervises other processes(child processes)
- Fits into a supervision tree
- Multiple supervision strategies: :one_for_one, :one_for_all, :rest_for_one, :simple_one_for_one

Distribution models

- **Distributed Elixir**
- **Socket based distribution**

Distributed Elixir

- programs are written to run on Elixir nodes
- A node is a self contained Elixir system with complete VM, memory and processes
- We can spawn a process on any node from the mesh
- Applications run in a trusted environment, on the same LAN and behind a firewall

Socket based distribution

- They can run in an untrusted environment
- Over TCP/IP
- Less powerful than Distributed Elixir but more secure

Take-aways

Processes are the building blocks: light weight processes, no shared state between actors, process linking and monitoring, all the communication happens by sending messages

OTP: patterns for handling process creation, supervision strategies and fault-tolerance support

Nodes: the only way to be really resilient is to have the application distributed via a mesh

There are no silver bullets :)

Thank you

Develop with passion!

Resources

https://pragprog.com/book/elixir/programming-elixir

http://elixir-lang.org/

http://elixirschool.com/

http://learnyousomeerlang.com/what-is-otp

http://learnyousomeerlang.com/the-hitchhikers-guide-to-concurrency

https://joearms.github.io/2013/04/05/concurrent-and-parallel-programming.html