High-availability Erlang from the trenches

Dominic Williams Fabrice Nourisson

Extreme Forge

8 november 2012

Sommaire

Extreme Forge

- Erlang and Agile (esp. eXtreme Programming)
- Training, consulting
- Developers for hire
- The Alonzo Quartet: a team of 4 experienced Erlang/Ember.js developers with an XP bent

`http://extremeforge.com`

Who we are

- We are:
  - Developers
  - Project managers
  - Consultants
  - Trainers
- With 15 years experience in:
  - Train control systems
  - Telecommunication systems
  - Web development

HA Erlang systems we've worked on

- Cellicium/Myriad USSD gateway/portal
  - 30 telco operators worldwide
  - Biggest deployments: 20 million users, 5000 MPS
  - 99,99% uptime
  - Monthly upgrades
- Initial architecture of MIG SMS gateway
- Corporama.com web site
- Telco operator call center web service

Underscore prefix considered harmful

Example

```
foo (X, _Args, _) ->
    case baz (X) of
        {ok, Result, _} ->
            Result;
        {error, _Args} ->
            error
    end.
```

Intent

Simplify code and avoid bugs

Underscore prefix considered harmful

Motivation

Unlike the anonymous variable (_), variables starting with an underscore (`_Foo`) are bound. They are usually used to silence the warning about unused variables. However, because they are bound, they can introduce bugs.

Recommendation

Never use the underscore prefix; only use the anonymous variable to ignore things.

Implementation

Patch proposed in erl_lint.erl to add warning.

Learn to use `gen_server` timeouts

Intent

Perform an action regularly when a `gen_server` is idle

Motivation

Many designs to achieve this are overly complicated:

- using a separate `gen_server`
- using timers
- ...

`gen_server` provides a little known timeout feature to achieve, very simply, this frequent design need.

Recommendation

Use the optional timeout in `handle_call` or `handle_cast` return tuples to perform regular idle actions

Learn to use gen_server timeouts

Example

```
handle_call (_, Msg, State) ->
    ...
    {reply, Reply, New_state, ?timeout}.

handle_info (timeout, State) ->
     {stop, normal, State}. % stops an idle process
```

Known uses

- Stopping an idle process
- Keeping a connection alive
- Closing an unused resource (file, socket...)
- Re-registering a lost worker

High-availability and *let it crash*

- Erlang provides everything to supervise and restart processes
- *In the small*, code is much cleaner and simpler if you let it crash
- In a larger sense, for very high availability, much care must be taken not to let the VM crash, or OS resources run out (disk space, file descriptors...)

Don't leak atoms

Example

```
fill () -> fill (0, init).

fill (N, _) ->
    Atom = list_to_atom (integer_to_list(N)),
    fill (N+1, Atom).

1> atom:fill().

Crash dump was written to: erl_crash.dump
no more index entries in atom_tab (max=1048576)
Aborted
```

Don't leak atoms

Intent

Prevent the atom table from filling up

Motivation

- The VM will crash if you use too many atoms (by default 1048576)
- Atoms are created in many ways:
  - hand-written code (modules, functions, intentional atoms)
  - generated code (e.g. ASN.1 compiler, yecc)
  - reading files (config, file:consult)
  - parsing (e.g. XML, JSON, ...)

Recommendation

Don't use `list_to_atom/1` and beware of libraries that do (e.g. xmerl). Use `list_to_existing_atom/1` or tag tuples with strings/binaries.

Use a fixed number of processes

Intent

Avoid running out of memory or overloading CPU

Motivation

- The VM will crash if it runs out of memory
- If system load goes up, unexpected things will start happening

Recommendation

Use a fixed number of processes (even connections, workers)

Always spawn fresh processes

Intent

Avoid unexpected bugs and leaks

Motivation

- Spawning and terminating Erlang processes has negligible cost
- Processes get dirty over time:
  - Leaks and old values in process dictionary
  - Unpurged message queue
  - Memory allocation/collection can be affected by history

Recommendation

Always spawn fresh worker processes; don't recycle them for several jobs

Use a job queue and a bounded number of workers

Intent

Control the load on the system

Motivation

- Use a fixed number of processes
- Always use fresh processes to perform work
- Have an easy way to balance load

Recommendation

Use a job queue and spawn fresh worker processes; put an upper bound on the number of workers.

Use a job queue and a bounded number of workers

Implementation

- Keep a queue of jobs to be performed
- Create fresh processes to perform work
- Limit the number (N * `erlang:system_info(schedulers)`)
- Make N configurable (and set according to load tests)
- Details depend on supervision needs, e.g.:
  - use `simple_one_for_one` supervised processes
  - spawn plain Erlang processes otherwise
- If workers are on multiple nodes, balance the load

Avoid records

Intent

Simplify hot code reloading

Motivation

- Records complicate hot code reloading
- Different versions of record are incompatible
- Public (API) records are the worst
- Process state (e.g. `gen_server`) can be handled by OTP but it is more complicated
- dict's are much simpler and more flexible

Recommendation

Avoid records, especially public records (included by several modules) and state records; prefer dicts, orddicts or proplists.

Code is data

Intent

Simplify hot code reloading by keeping data in Erlang code

Motivation

- External files (config, data, templates...) affect the behaviour of the system
- Changes are not handled as well as hot code reloading
- Because Erlang code can be reloaded, there is no real need for putting anything in files
- Config files are a legacy from the days when code was harder to upgrade than text files

Recommendation

Use code (Erlang modules) for everything, including configuration

Miscellaneous recommendations

- Systematically perform serious performance tests and measures: load, endurance, capacity, stress, ... This requires a dedicated platform and serious effort.
- Systematically test upgrading (hot code reloading) before doing it on live system
- Don't log debug info (only customer's auditing/history)
- Become familiar with tracing, dbg etc.
- Become familiar with Erlang's system limits (cf. doc)
- File system: log rotation, ...

Questions?

mailto:contact@extremeforge.com