HA Erlang depuis les tranchées

Dominic Williams   Fabrice Nourisson

Extreme Forge

30 november 2012

Sommaire

Intro

HA Erlang patrons et idiomes

- Conseils de codage
- Mettre des limites
- Mise à jour à chaud du code
- Divers

Q&A

Extreme Forge

- Erlang and Agile (part. eXtreme Programming)
- Formation, conseil
- Mission de développement
- The Alonzo Quartet: une équipe de 4 développeurs Erlang/Javascript expérimentés utilisant XP

`http://extremeforge.com`

Qui sommes nous

- Nous sommes :
  - Développeurs
  - Chefs de projet
  - Consultants
  - Formateurs
- Avec 15 ans d'expérience dans :
  - Systèmes ferroviaires
  - Systèmes de Télécommunication
  - Développement Web

Système HA en Erlang sur lesquels nous avons travaillé

- Cellicium/Myriad USSD gateway/portail
  - 30 opérateurs de télécom dans le monde entier (#123# chez OrangeFrance)
  - Déploiement le plus important: 20 millions d'utilisateurs, 5000 MPS
  - 99,99% uptime
  - Mise à jour mensuelle
- Architecture initiale de la gateway MIG SMS
- Site web Corporama.com
- Web service du call center d'un opérateur de télécom

Préfixer les variables par un souligné est nocif

Example

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

Intention

Simplifie le code et évite les bugs

Préfixer les variables par un souligné est nocif

Motivation

A la différence des variables anonymes (`_`), les variables commençant par un souligné (`_Foo`) sont liées. Elles sont utilisées pour supprimer le warning sur les variables inutilisées. Mais comme elles sont liées, elles introduisent des bugs.

Recommendation

Ne jamais utiliser le préfixe souligné; utiliser seulement la variable anonyme pour ignorer des choses.

Implementation

Amélioration proposée dans erl_lint.erl pour ajouter un warning.

Apprendre à utiliser les timeouts des `gen_server`

Intention

Exécuter une action régulière quand un `gen_server` est au repos

Motivation

Beaucoup d'architectures trop compliquées pour y arriver:

- utiliser un autre `gen_server`
- utiliser des timers
- ...

`gen_server` fournit une fonctionnalité timeout peu connue permettant très simplement de réaliser ce besoin très fréquent.

Recommendation

Utiliser le timeout optionnel dans le tuple de retour de `handle_call` ou `handle_cast` pour déclencher au repos des actions régulières

Apprendre à utiliser les timeout des gen_server

Example

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

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

Retours d'expèriences

- Arrêter un processus au repos
- Conserver une connexion active
- Fermer une ressource plus utilisée (fichier, socket ...)
- Ré-enregistrer un worker perdu

Haute Disponibilité et *let it crash*

- Erlang fournit tout pour superviser et redémarrer les processus
- En résumé, le code est plus clair et plus simple si vous le laissez crasher
- Pour la très haute disponibilité, vous devez faire attention de ne pas crasher la VM, ou arriver à la limite des ressources de l'OS (espace disque, CPU)

Attention à la fuite des atomes

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
```

Attention à la fuite des atomes

Intention

Empêcher le remplissage de la tables des atomes

Motivation

- La VM crash si il y a trop d'atomes (par déf. max 1048576)
- Les atomes sont créés de plusieurs façons:
  - à la main dans le code (modules, fonctions, atomes)
  - code généré (par ex: compilateur ASN.1, yecc)
  - lecture de fichiers (config, file:consult)
  - parsing (par ex. XML, JSON, ...)

Recommendation

Ne pas utiliser `list_to_atom/1` ou `binary_to_atom/1` et attention aux librairies (par ex. xmerl). Utiliser `list_to_existing_atom/1`, `binary_to_existing_atom/1` ou utiliser les tuples taggués avec strings/binaries.

Utiliser un nombre fixe de processus

Intention

Éviter de saturer la mémoire et le CPU

Motivation

- La VM crashera si elle est à court de mémoire
- La création de processus échoue si il y trop de processus (par déf. max 32768)
- Si le système est surchargé, des choses inattendues vont apparaître

Recommendation

Utiliser un nombre fixe de processus (même pour les connexions, workers)

Toujours démarrer des nouveaux processus

Intention

Empêche les bugs étranges et fuites mémoires

Motivation

- Démarrer et terminer des processus Erlang a un coût négligeable
- Les processus pourrissent avec le temps :
  - Fuites mémoire et anciennes données dans le dictionnaire de processus
  - Queue des messages non vidée
  - L'allocation de mémoire peut être affectée avec le temps

Recommendation

Toujours démarrer des nouveaux processus; Ne les recycler pour plusieurs choses

Utiliser une queue de tâches et un nombre limité de workers

Intention

Contrôler la charge du système

Motivation

- Utiliser un nombre fixe de processus
- Toujours utiliser des nouveaux processus pour effectuer le travail
- Avoir un moyen facile de faire de l'équilibrage de charge

Recommendation

Utiliser une queue de tâches et toujours démarrer des nouveaux processus; Définir le nombre maximum de processus.

Utiliser une queue de tâches et un nombre limité de workers

Implementation

- Garder une queue de tâches à effectuer
- Créer des nouveaux processus pour effectuer le travail
- Limiter le nombre (N * `erlang:system_info(schedulers)`)
- Rendre N configurable (et le définir en fonction des tests de charge)
- Les détails d'implémentation dépendent des besoins de supervision, par ex. :
  - utiliser `simple_one_for_one` pour des processus supervisés
  - démarrer de 'simple' processus Erlang sinon
- Si les workers sont sur plusieurs noeuds, partager la charge

Éviter les records

Intention

Simplifie le rechargement à chaud du code

Motivation

- Les records complexifient le rechargement à chaud du code
- Les différentes versions du records sont incompatibles
- Une interface public (API) utilisant les records est mauvaise
- L'état du processus (par ex. `gen_server`) peut être traités par OTP mais cela est plus compliqué
- les 'dicts' sont plus simples et plus flexibles

Recommendation

Éviter les records, particulièrement les records publics (inclus dans plusieurs modules) et les records d'états; préférer les dicts, orddicts ou proplists.

Le code est de la donnée

Intention

Simplifier le rechargement du code à chaud en mettant les données dans le code Erlang

Motivation

- Les fichiers externes (config, données, canevas...) affectent le comportement du système
- Les modifications de fichiers ne sont pas aussi bien gérées que la mise à jour à chaud
- Les fichiers de configuration sont un héritage d'une époque où la mise à jour du code était plus difficile que la mise à jour de fichiers textes

Recommendation

Utilise le code (modules Erlang) pour tout, configuration inclus

Recommandations diverses

- Effectuer systématiquement des 'vrais' tests de performance et de mesure: charge, endurance, capacité, stress, ... Cela demande une plate-forme dédiée et beaucoup d'effort
- Faire des tests systématiques de la mise à jour avant de la faire sur le système en production
- Ne pas mettre dans les logs des informations de debug (Uniquement des informations utiles pour le client : audit/historique)
- Devenir familier avec le tracing, debug etc.
- Devenir familier avec les limites du système Erlang (cf. doc)
- Système de fichier : rotation des logs, . . .

Questions?

mailto:contact@extremeforge.com