

Erlang における最適な監視ツリーの自動生成法

1265101 佐々木 勝一 【ソフトウェア検証・解析学研究室】

Automatic Generation of an Optimal Supervision Tree in Erlang

1265101 SASAKI, Shoichi 【Software Verification and Analysis Lab.】

1 はじめに

Erlang とは並行指向の関数型プログラミング言語であり、OS スレッドより軽い軽量プロセスを標準で備えている。Erlang の大きな特徴として、想定外のエラーが発生した軽量プロセスはそのまま終了させて再起動する、という考え方のエラーハンドリングが主流な点が挙げられる。Erlang は、この考え方を実現する仕組みとして監視ツリーを提供している。監視ツリーの設定により、いくつかの制限はあるが、プロセス終了時に再起動されるプロセスらを制御できる。監視ツリーを作成する際の要求として、再起動中のプロセスはプログラムを実行できないから、一度に再起動されるプロセスの数を最小限にしたい。一方で、依存するプロセスが終了したプロセスは処理の継続が困難と判断できるから、終了したプロセスに依存する全てのプロセスは再起動したい。監視ツリーでは、仕様由来の制限により、どうしても無関係なプロセスを再起動せざるを得ない場合があり、これらの要求を満たす監視ツリーの作成は容易ではない。通常はこれらの要求を満たす監視ツリーをプログラマが注意深く作成するが、プロセスの数が増えてくると困難になる。

この問題解決のため、Neykova ら [1] は、プロセス間の通信の向きや順序を表現可能な型システムを利用し、従来の監視ツリーとは異なる仕組みで最小限のプロセスのみ再起動する手法を提案した。しかし、この手法はプロセスの定義方法に強い規約を設けており、従来の Erlang システムへの適用は難しい。

本研究では、従来の Erlang システムに適用可能なプロセスの再起動法の実現を目指し、最適な監視ツリーの自動生成法を提案する。最適な監視ツリーを生成するために解くべき問題を定式化し、その問題を解くアルゴリズムを開発した。提案アルゴリズムは、病的なケースに対しては指数関数時間かかるが、現実的な多くの場合に対しては十分高速に動作することを実験的に示した。

2 Erlang

Erlang には、Java の継承に似る、特定の振る舞いを行う軽量プロセスの実装を支援する機構がある。その一種が `gen_server` とスーパーバイザである。特に、スーパーバイザは軽量プロセスを監視・再起動するプロセ

スを実装でき、一般的にスーパーバイザを用いて監視ツリーが作られる。監視ツリーにおいて、`gen_server` は葉、スーパーバイザは内部節点に相当する。なお、スーパーバイザの子プロセス間には順序が定められる。

各スーパーバイザに設定される再起動戦略によって、再起動されるプロセスが決定される。提案手法がサポートする再起動戦略は、`one_for_one`, `one_for_all`, `rest_for_one` の 3 つである。`one_for_one` は終了した子プロセスのみを再起動する。`one_for_all` は終了した子プロセスだけではなく、他の全ての子プロセスもまた再起動する。`rest_for_one` は終了した子プロセスと「残り」の子プロセスを再起動する。「残り」の子プロセスとは、終了した子プロセスより順序が後のプロセスを指す。なお、スーパーバイザが再起動されるときは、そのスーパーバイザの子プロセスが再帰的に再起動される。

3 問題の定式化

$\mathbb{N} = \{1, 2, \dots\}$ は自然数の集合とし、有限集合 A の要素数を $|A|$ と書く。また、任意の有向グラフ $G(V, E)$ と頂点 $v, u \in V$ に対して、 v から u に到達可能であることを $v \rightsquigarrow_G u$ と書く。

有向グラフ $G_d(V_g, E_g)$ を依存関係グラフと呼ぶ。 V_g は `gen_server` の有限集合であり、 $E_g \subseteq V_g \times V_g$ は `gen_server` 間の依存関係である。 $(g_1, g_2) \in E_g$ は「 g_1 は g_2 に依存する」ことを表す。

依存関係グラフ $G_d(V_g, E_g)$ に対し、後述の制約を満たす 6 字組 $T(V_c, V_s, r, \pi, ord, stg)$ を部分監視ツリーと呼ぶ。 $V_c \subseteq V_g$ は葉の `gen_server` の有限集合、 V_s はスーパーバイザの有限集合、 $r \in V_s$ は根、 $\pi : (V_c \cup V_s \setminus \{r\}) \rightarrow V_s$ は親、 $ord : (V_c \cup V_s \setminus \{r\}) \rightarrow \mathbb{N}$ は兄弟間の順序、 $stg : V_s \rightarrow \{\text{of}, \text{ofa}, \text{rfo}\}$ はスーパーバイザの再起動戦略を表す。なお、 $V_c = V_g$ である部分監視ツリーは単に監視ツリーと呼ぶ。 T は次の制約を満たす：

- $V_c \cap V_s = \emptyset$
- $(V_c \cup V_s, r, \pi)$ は根付き木
- 任意の $s \in V_s$ と $v, u \in \text{child}(T, s)$ に対して、 $v \neq u$ ならば $ord(v) \neq ord(u)$
- 任意の $v, u \in V_c$ に対して、 $v \rightsquigarrow_{G_d} u$ ならば $v \in \text{restart}(T, u)$

ただし, $child(T, v)$ は v の子プロセスの集合を表し, $child(T, v) = \{u \mid \pi(u) = v\}$ で定義される. また, $restart(T, v)$ は v の終了時に再起動されるプロセスの集合を表す. ここで, $sibling(T, v) = child(T, \pi(v))$, $follow(T, v) = \{u \in sibling(T, v) \mid ord(v) < ord(u)\}$ とすると, $restart(T, v \in V_c \cup V_s)$ は次の制約を満たす最小の集合である:

1. $v \in V_c$ ならば $v \in restart(T, v)$
2. $v \in V_s$ ならば $\bigcup_{u \in child(T, v)} restart(T, u) \subseteq restart(T, v)$
3. $stg(\pi(v)) = ofa$ ならば $\bigcup_{u \in sibling(T, v)} restart(T, u) \subseteq restart(T, v)$
4. $stg(\pi(v)) = rfo$ ならば $\bigcup_{u \in follow(T, v)} restart(T, u) \subseteq restart(T, v)$

コストは $cost(T) = \sum_{v \in V_c} |restart(T, v)|$ と定義される.

以上の定義を用いて, 最適監視ツリー問題は, 依存関係グラフ (V_g, E_g) が入力され, コストが最小の6字組 $(V_g, V_s, r, \pi, ord, stg)$ を出力する問題と定義される.

4 最適監視ツリー問題を解くアルゴリズム

提案アルゴリズムでは, vertex splitter と呼ばれる「良い」性質を持った頂点集合を求めることが重要である. 連結な有向非巡回グラフ $G(V, E)$ に対する vertex splitter $S \subseteq V$ は, 1) 任意の $s \in S$ と $v \in V$ に対して, $s \rightsquigarrow_G v$ ならば $v \in S$, 2) $S \neq V$ ならば $|components(subgraph(G, V \setminus S))| \geq 2$, の2つの制約を満たす頂点集合と定義される. ただし, 任意の有向グラフ $G(V, E)$ に対して, $components(G)$ は辺の向きを無視したときの連結成分の集合を表し, $subgraph(G, U \subseteq V)$ は頂点集合が U である G の誘導部分グラフを表す.

提案アルゴリズムは, 前処理として, 入力 G_d の強連結成分グラフ $G_D(V, E, w)$ を求める. ただし, $w: V \rightarrow \mathbb{N}, w(v) = |v|$ である. 提案アルゴリズムは, G_D を入力として, 1) まず, G_D の全ての極小の vertex splitter の集合 $\mathbb{S}$ を求め, 各 $S \in \mathbb{S}$ に対して, 2) 葉として S のみを含み, ofo のスーパーバイザを含まず, スーパーバイザを除く兄弟は同じ強連結成分に含まれる頂点のみであるような監視ツリー T を求め, 3) $subgraph(G_D, V \setminus S)$ の各連結成分に対し再帰して複数の監視ツリーを得て, 4) その全ての監視ツリーを子とする, ofo のスーパーバイザが根の監視ツリー T' を作り, 5) T の最も右下のスーパーバイザの子の末尾に T' を結合して T'' を作る. 6) 各 S に対する T'' の中で最もコストが小さいものを出力とする. このアルゴリズムは正当性と最適性どちらも満たす. 最適性の証明の主なアイデアは, ある極小の vertex splitter と最適な監視ツリーは構造が一致するから, 全ての極小の vertex splitter に対して求めた監視ツリーの中に最適な監視ツリーが存在する, というものである.

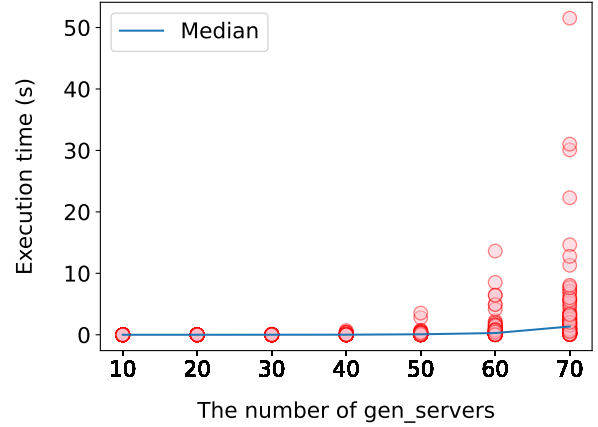


図1 グラフの頂点数と辺数が等しい場合の実行時間

上記のアルゴリズムでは全ての極小の vertex splitter の集合を求めるが, これを多項式時間で解くアルゴリズムはまだ発見されていない. しかし, 最適監視ツリー問題を解く場合は, 各極小 vertex splitter S ではなく vertex splitter の制約1を満たす空でない S の部分集合 (の任意の一つ) を求めれば十分であることを証明し, これは多項式時間で解けることを示した.

5 評価

入力のグラフの頂点数を V とすると, 提案アルゴリズムは, 病的なケースに対しては $O(V^2 2^V)$ の計算時間がかかることがわかっている. しかし, 現実的な多くの場合のグラフに対しては, 十分高速で動作することを示すための実験を行なった. 実験では, 頂点数と辺数を変動させたグラフをランダムに生成し, そのグラフを提案アルゴリズムに入力して実行時間を計測した. 実験は各頂点数に対して100回試行した. 図1は頂点数と辺数が等しい場合の結果である. この実験より, $V = 70$ の場合でも, 実行時間の中央値は1.3415秒であった. なお, 辺数が頂点数より多い場合は, G_d の強連結成分が増えて G_D の頂点数が少なくなり, 実行時間はより短くなる. 2024年1月19日時点でGitHubに公開されている999個のErlangシステムを調査したところ, gen_serverの数が70個以下のシステムは約99.6%だったため, 提案アルゴリズムは十分高速に動作すると考える.

6 おわりに

本研究では, 最適な監視ツリーの自動生成法として, 定式化した問題を解くアルゴリズムを提案した. 提案アルゴリズムは, 現実的な多くの場合に対しては十分高速に動作することを実験的に示した.

参考文献

- [1] Rumyana Neykova and Nobuko Yoshida. Let it recover: multiparty protocol-induced recovery. In *CC'17*, pp. 98–108, 2017.