Rust を用いて OTP アプリケーションを構築するための フレームワークの開発

1265093　岡本 怜士　【 ソフトウェア検証・解析学研究室 】

Development of a Framework for Implementing OTP Application with Rust

1265093　Okamoto Reiji　【 Software Verification and Analysis Lab. 】

1　はじめに

プログラミング言語 Rust は，並行性の実現に対して OS スレッドもしくはグリーンスレッドの 2 通りの方法をサポートしている．グリーンスレッドを用いる方法は，OS スレッドと比較して大量のタスクを処理する場合でも CPU やメモリのオーバヘッドを小さく抑えられる．しかし，例えば以下のような要因で設計が複雑になる．

1. 低水準の関数を複雑に組み合わせて，目的の機能を実装する必要がある
2. 各タスクをどのスケジューラが管理するかをプログラマが決める必要がある
3. あるタスクで発生したパニックがスケジューラまで伝播すると他のタスクを道連れにする

近年，グリーンスレッドの機能の抽象化をはかる非標準フレームワーク Actix[1]が開発され，上記の問題点 1 は大幅に緩和されたが，2，3 は依然解決していない．

そこで我々は，並行指向の関数型プログラミング言語，およびその標準ライブラリ等のセットである Erlang/OTP に注目した．OTP は，大量のプロセスが並行動作し，かつ耐障害性を満たすシステムのための設計原則を提唱しており，そのようなシステムを構築するためのライブラリを提供している．また，Erlang/OTP は機械語関数を呼び出すための機構を標準でサポートしている．これらの機能を応用することで，Rust のみを用いて OTP アプリケーションが構築できる，つまり耐障害性を満たし，かつ大量のプロセスが並行動作するアプリケーションを Rust で容易に構築できると考えられる．そこで本研究では，Rust を用いて Erlang VM 上で動作する OTP アプリケーションを構築するフレームワークを提案する．提案システムの評価のため，ベンチマークプログラムを作成して提案フレームワーク・Actix・OS スレッドをそれぞれ用いた場合の性能を比較する．また，アプリケーションの構築容易性や耐障害性に関する検討を行う．

[1] https://github.com/actix/actix

2　提案フレームワークの概要

OTP アプリケーションはワーカ・スーパバイザの考え方に基づく監視ツリーと呼ばれる構造を持つ．ワーカは実際の計算を行うプロセスで，スーパバイザは子プロセスを監視し，必要に応じて再起動する役割を持つ．

監視ツリーを構成する各プロセスの実装方法はイベント駆動型のプログラミングモデルに基づいており，プロセスの起動，メッセージの受信などをきっかけにユーザが定義した Rust 関数がコールバックされる．この手法を実現するため，以下の二つのツールを開発した．

2.1　ビルドツール cargo-otp

ユーザが記述した Rust プログラムから OTP アプリケーションをビルドし，実行するためのツール cargo-otp を開発した．Rust 関数を呼び出すための Erlang モジュールを自動で展開することでユーザが Erlang コードを書かなくても良いように設計した．

2.2　ライブラリ otp_wrap

内部システムで二つの言語を用いるため，Rust コンパイラは Erlang が要求するコールバック関数が適切に定義されているかを検査できない．そこで，実装されていないコールバック関数を検知してコンパイルを拒否するためのライブラリを開発した．

3　性能評価実験

比較対象は，提案フレームワーク，Actix，OS スレッド（スレッドプールは用いない）の 3 つである．アクターベンチマークスイート Savina[1] を参考にそれぞれの手法でベンチマークプログラムを 4 種類実装した．評価結果を図 1 に示す．実験環境は，Intel(R) Xeon(R) E-2378 CPU @ 2.60GHz 8C/16T，UDIMM 64GB，Ubuntu 22.04.2，rustc 1.75.0，Erlang/OTP 26 である．

提案フレームワークは，Big ベンチマークだけ大幅に性能が低下し，他は Actix を用いる場合と同程度であり，OS スレッドよりは優れていた．Ping Pong と Thread

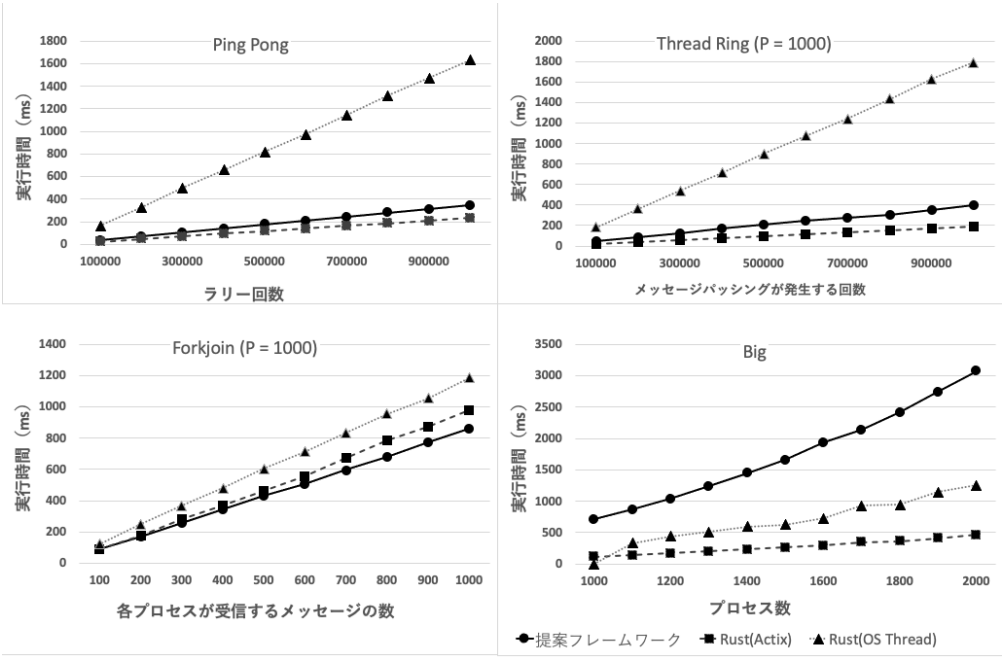

図 1　ベンチマーク評価結果

Ring は OS スレッドより大幅に優れていた．

Big ベンチマークの実行状況を分析したところ，受信したメッセージを保存しておくためのメールボックスに対する競合の影響が計測された．このことから，あるアクターが同時に複数のメッセージを受信しなければならないような状況では提案フレームワークの性能が劣化するものと考えられる．Erlang を含むいくつかの言語についてメッセージパッシングの性能を調査した研究 [2] では，レイテンシは小さいと報告されており，上記のような傾向は見られない．しかし，[2] の調査は機械語関数が呼び出される際の影響を考慮していないため，今後，Rust 関数を実行しているプロセスのメールボックスに対する競合の影響を詳細に調査する必要がある．

4　構築容易性・耐障害性に関する検討

4.1　ユーザが記述しなければならない処理

グリーンスレッド API は低水準で，直接使ってアプリケーションを実装するのはコストが大きい．提案フレームワークはこの負担を大幅に緩和している．Actix を使えば，提案フレームワークと同レベルの抽象度でグリーンスレッドの機能を利用できるが，コールバック関数の実装の他に，ランタイムシステムやスケジューラの起動などを明示的に記述しなければならない．提案フレームワークではコールバック関数のみを記述すればよく，残りの処理はビルドツールが提供する CLI を利用するだけで Erlang VM が自動的に行ってくれる．

4.2　耐障害性

Actix も OTP のスーパバイザに似た機能をサポートしている．しかし，単にアクターが停止すれば何かをする（例えば再起動する）ように拡張するだけで異常終了には対処できない上に，同じスケジューラ上のタスクを全て巻き添えにしてクラッシュする．したがって，エラーハンドリングを適切に行い，スケジューラに伝搬しないように注意深くコードを書かなければならない．提案フレームワークは，異常を検知すればすぐにクラッシュさせるように設計しても，スーパバイザによって再起動させられ，他のタスクには影響を及ぼさない．

5　まとめ

本研究では，Rust を用いて Erlang VM 上で動作するアプリケーションを構築するフレームワークを提案した．Actix と比較して，OTP が提供する高水準な機能群を利用できるようになったことで耐障害性を満たし，大量のプロセスが並行動作するシステムを開発する際にユーザの負担を軽減することに成功したと言える．

ただし，現状ではある程度 Erlang のデータ型に関する知識を持っていることを想定している．不適切な型の値を返す Rust 関数を実装するとプロセスがクラッシュする危険があるため，Erlang のデータ型の知識がなくても利用できるように拡張することが今後の課題である．

参考文献

[1] S. M. Imam and V. Sarkar. Savina — an actor benchmark suite: Enabling empirical evaluation of actor libraries. In *AGERE! '14*, pp. 67–80, 2014.

[2] I. Valkov, N. Chechina, and P. Trinder. Comparing languages for engineering server software: Erlang, Go, and Scala with Akka. In *SAC '18*, pp. 218–225, 2018.