Ractorを用いたRubyの並列処理性能評価と静的解析ツールによるコード支援

成蹊大学理工学部情報科学科　ソフトウェア研究室　S202148　柳澤快

1. 研究背景

　Rubyは1995年に一般公開されたオブジェクト指向スクリプト言語である。Rubyは他の言語(Perl、Smalltalk、Eiffel、Ada、Lisp)の影響を受けて設計させている。

　Rubyは多くの支持を集めているが、こうした成長はRubyで書かれたソフトウェア、WebアプリケーションフレームワークRuby on Railsの人気に起因している。Ruby on RailsはDavid Heinemeier Hansson氏により2004年に誕生し、Webアプリケーションを手軽に開発できるように設計されている。Ruby on Railsは最善の開発方法は1つであるという大胆な判断に基づいて設計されている。このRails Way、すなわちRailsというレールに乗って開発することで開発の生産性が向上し、開発する際のコード量が少なく済むことで保守しやすくなり、バグも減らすことが可能になる。このことからRuby on Railsは、開発工数を抑えられ、比較的短期間で目的のサービスやアプリ開発の実現を最大の特徴としている。このようなRuby on Railsの利点から、RubyはSaaS企業やスタートアップ企業をはじめとして、数多くの採用例がある。

　また、近年、コンピュータの性能はマルチコアプロセッサの普及により向上している。それに伴い、ソフトウェアの並列処理能力が重要となっている。並列計算機上で複数の処理を同時に実行するために、多くのプログラミング言語では複数スレッドを同時実行させることができる。しかし、Rubyのスレッドはグローバルインタープリターロックによって同時に実行可能なスレッドは1つのみとなっている。これにより、スレッドを使う限り、通常の方法では並列プログラムをRubyで記述することはできない。

　Ractorの登場により異なるRactor間でスレッドを並列に実行することが可能になった。本研究では、Ractorで記述されたRubyの並列処理の性能評価を行うことにより、Ractorの最適な記述方法を明らかにし、静的解析ツールによるRactorによる記述への支援を目指す。

1. Ractorの概要

　本章では、並行・並列処理を可能にするActor Modelの抽象化であるRactorについて説明する。

* 1. Ractorとは

　Ractorとは、2020年にリリースされたRuby3.0で導入された並行・並列処理を可能にする機構であり、スレッドセーフな並列実行を提供するRubyのActor Modelの抽象化である。Ractorは元々、Guildという名前で開発が始められており、2020年にRactorという名前に変更された。

　Actor Modelは、1973年に米国マサチューセッツ工科大学のCarl Hewitt氏によって発表された並列計算の数学的モデルの一種である。

* 1. Ractorの生成

　Ruby処理系を起動すると、1つのRactorを作成する(これをMain Ractorと呼ぶ)。同一Ractorは最低1つのスレッドを持つ。同一Ractor内の複数スレッドはグローバルインタープリターロックにより同時には実行されない。Ractorの生成には、class Ractorを用いる。

r1 = Ractor.new { p “I am in r1” }

r2 = Ractor.new { p “I am in r2” }

r1.take

r2.take

　この例では、r1、r2という2つのRactorを生成している。Ractorを生成すると、暗黙にスレッドも生成され、別Ractorに所属するスレッドであるため、並列処理を実現することができる。Ractor#takeメソッドで対象Ractorが終了するまで待ち、メッセージを受け取る。r2がr1より早く処理が終了した場合は、”I am in r2”が先に出力される。

* 1. 共有可能オブジェクト

　Ractorは異なるRactor間とのオブジェクトの制限している。データレースやレース状態などのスレッド安全性をユーザーが意識することなく使用できるようにするためである。

　共有可能オブジェクトは次の3種類となっている。

* 不変オブジェクト
* クラス・モジュールオブジェクト
* その他の特殊なオブジェクト(Ractorオブジェクト自体)

　これら以外のオブジェクトはすべて共有不可能なオブジェクトである。さらに、Ractor間で共有可能なオブジェクトであるかを調べるためにRactor.shareable?メソッドが提供されている。

* 1. Ractor間通信

　Ractorでは共有メモリモデルをとらないため、メッセージの伝達手段が必要になる。Ractor間でメッセージを送受信する方法として2種類提供されている。

* + 1. push型通信

　送信者が受信者を知っている場合のメッセージ交換手法を示す。Ractor#sendによって、対象Ractorへメッセージを送信する。Ractor.receiveによって、自Ractorへ届いたメッセージを受信する。

r = Ractor.new { p Ractor.receive }

r.send(“string\_object”)

r.take

　この例では、Main Ractorから子Ractorであるrへオブジェクト(“string\_object”)を送り、Ractor.receiveで受け取ったオブジェクトをpメソッドで標準出力している。

* + 1. pull型通信

　受信者が送信者を知っている場合のメッセージ交換手法を示す。Ractor.yield(obj)によって送信されたかのように振る舞い、引数objが外部のRactorに送信される。Ractor#takeで対象Ractorのメッセージを受け取る。

r1 = Ractor.new 10 do |arg|

Ractor.yield arg

end

r2 = Ractor.new r1 do |arg|

arg.take

end

p r2.take

　この例では、Ractor.yieldとRactor#takeの使用例を示している。10で初期化したRactor r1とr1で初期化したr2を生成し、r1のブロックに引数argとして10が渡され、argを送信する。r2のブロックに引数argとしてr1が渡され、argを受け取る。r2.takeで受け取ったオブジェクト(“10”)を標準出力する。

1. 実験

　バブルソートを用いてRactorの実験を行った。

1. Rubocopの概要
2. まとめ

参考文献

1. Ruby Programming Language: <https://www.ruby-lang.org/ja/>
2. Ruby on Rails ガイド: <https://railsguides.jp/>
3. SIOS Tech.Lab.「アクターモデル」による並列処理プログラミング入門: <https://tech-lab.sios.jp/archives/8738>
4. Ruby-Doc.org.Ractor: <https://ruby-doc.org/core-3.0.0/Ractor.html>
5. Documentation for Ruby. class Ractor: <https://docs.ruby-lang.org/en/master/Ractor.html>
6. Cookpad Developers Blog. Rubyの並列並行処理のこれまでとこれから: <https://techlife.cookpad.com/entry/2023/08/31/152511>
7. 笹田耕一. Ruby3.0のRactorについて: <https://www.atdot.net/~ko1/activities/2021_ginzarails.pdf>
8. 笹田耕一. Ruby向け並列化機構Guildの試作. 情報処理学会プログラミング研究会(2018年8月1日)