JuliaとActorを用いた強化学習フレームワークの提案

中田秀基

産業技術総合研究所 {デジタルアーキテクチャ,人工知能}研究センター

概要

並列強化学習には複雑な並行制御が必要となり、単純なFork-Join型の並列計算にはそぐわない。本発表ではJulia言語を対象としてActorを導入し、その上に汎用性の高い教科学習フレームワークを構築する。

本研究はJSPS科研費JP19K11994の助成を受けたものです

背景

- 強化学習には膨大な計算が必要
  - –並列計算が不可欠
  - –複雑な同期制御
  - –並列計算機の利用が不可欠
- Python
  - - 機械学習でドミナント
  - - 高速化は試みられているが低速
- Julia言語
  - –高速、機械学習分野で利用が広がる
- Julia 向けに分散フレームワークがほしい

目的と成果

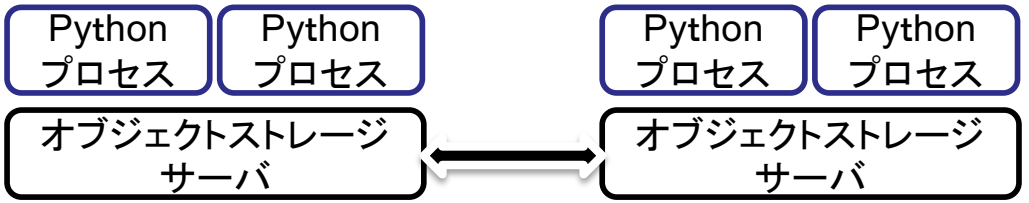
- 目的
  - 利用者が並列分散を意識せずにプログラミング可能な強化学習フレームワークを提供
- 成果
  - Julia 言語上にActor機構を提案(既発表)
  - Actor機構を用いた強化学習フレームワークを設計、プロトタイプを実装
  - 並列実行の効果をOpenAI gymで実験

Python向け分散フレームワークRay

- Berkeley riselab のPython用並列分散フレームワーク
  - 強化学習を対象
- Actorとして記述
  - remote をつけたクラスをリモートでActorとして生成
- 同期はfuture

```
import ray
ray.init()

@ray.remote
class Counter(object):
    def __init__(self):
        self.n = 0
    def increment(self):
        self.n += 1
        return self.n

c = Counter.remote()
future = c.increment.remote()
print(ray.get(future))
```

- ノード間通信は共有オブジェクトストレージを介する
- ノード内通信は共有メモリ



- 問題点：
  - Python – GIL-スレッドがまともに動かない, そもそも低速
  - 共有オブジェクトストレージが比較的低速

Julia言語

- 高速なスクリプト「風」言語
  - LLVMを用いたJITコンパイル
  - 実行時に実際に呼び出された型に特化したコードを動的に生成
  - 型を静的に指定することも可能
  - Cに匹敵する(?)実行速度
- 3種類の並行/並列実行をサポート
  - コルーチン：libuv を使用。I/Oでブロックすると他のコルーチンに制御が移る
  - マルチスレッド: OSスレッドを使用
  - マルチノード: SSH/MPIを利用した複数ノードでの起動をサポート

Julia言語(2)

- CLOS的なOOP – struct と Generic function で構成
  - クラスに属する「メソッド」がない

Python

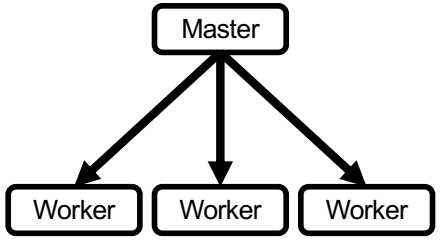
```
class Shape:
  pass
class Circle(Shape):
  def __init__(self, radius):
    self.radius = radius
  def area(self):
    return self.radius * self.radius\
           * 3.14

class Rectangle(Shape):
  def __init__(self, width, height):
    self.width = width
    self.height = height
  def area(self):
    return self.width * self.height

print(Circle(10).area())
print(Rectangle(10,20).area())
```

Julia

```
abstract type Shape end

struct Circle <: Shape
    radius::Real
end

struct Rectangle <: Shape
    height::Real
    width:: Real
end

area(c::Circle) = c.radius * c.radius * pi
area(r::Rectangle) = r.height * r.width

area(Circle(5.0))
area(Rectangle(10,20))
```

Juliaのマルチノード並列: Distributed.jl

- RPC – 関数の実行ノードを指定
  - 引数は自動的に転送される
  - 呼び出しは即時にリターン
  - 返り値はfuture – 同期機構
  - futureの値をfetchしようとした時点で、まだ計算が終了していなければそこでブロック



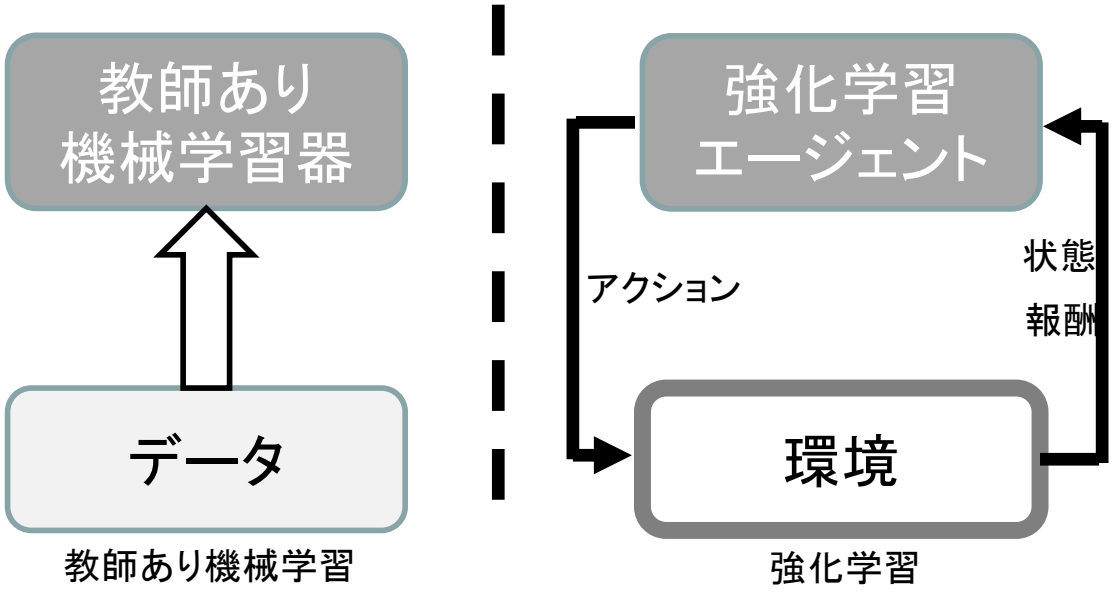

```
future = @spawnat 2 f(X)
value = fetch(future)
```

- リモートチャンネル
  - リモートノード上のチャンネルのグローバルな参照
  - 直接書き込み可能
- 問題点
  - リモートノード上の状態を管理する方法がない
  - グローバル変数に書き込むことは可能

Actor

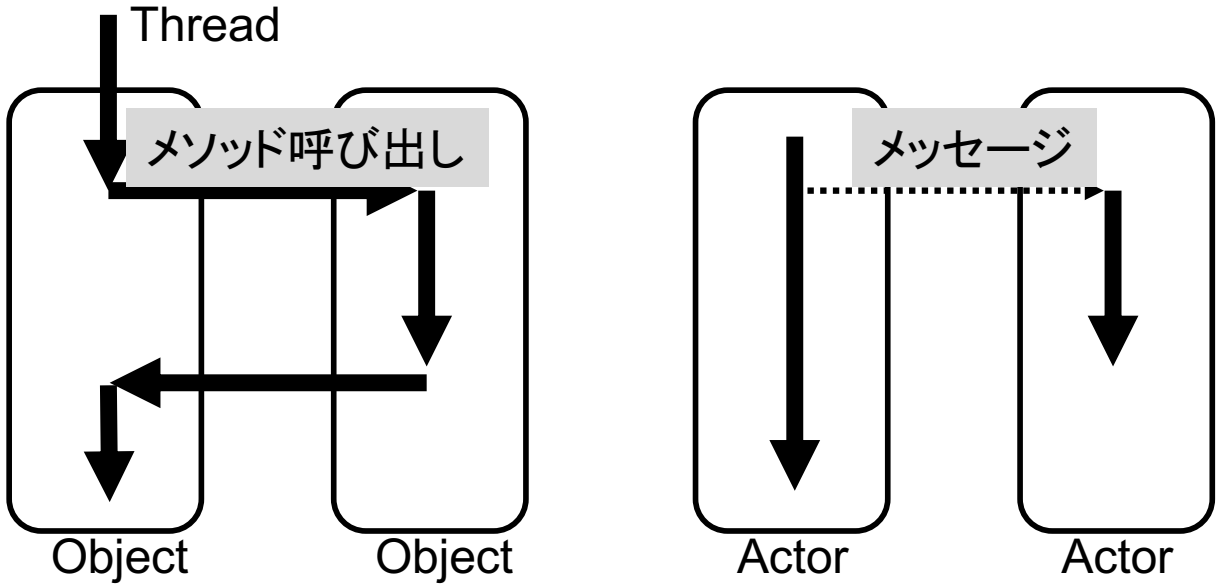
強化学習

- 教師あり、教師なしに並ぶ、機械学習の1ジャンル
- 報酬が遅延するため学習が困難
- 試行錯誤を通じてエージェントが環境に適応
  - 環境との相互作用が必要
  - 環境の更新が高価な場合も



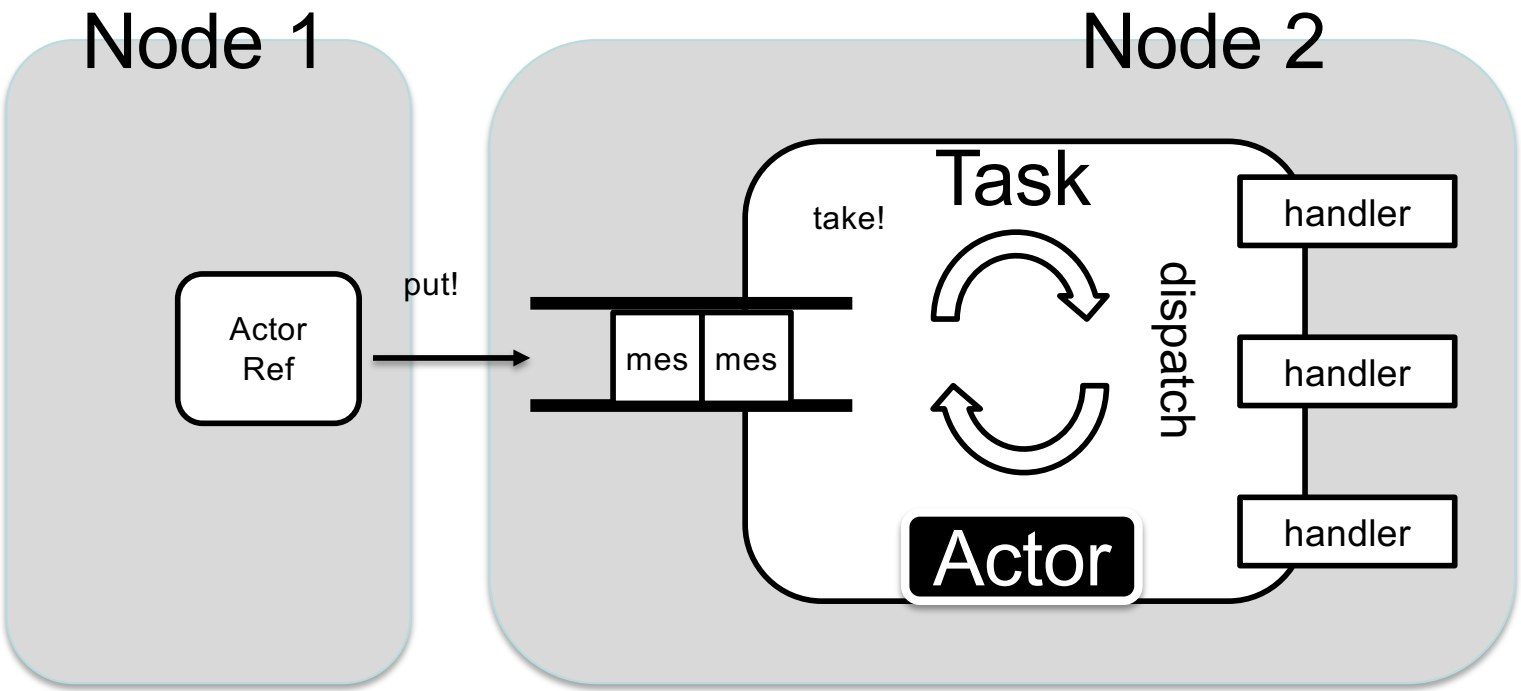
Actor

- 状態を持つ実行主体がメッセージを受信して処理
  - 状態と実行主体が1対1に対応
    - Object ＋ Threadとは本質的に異なる
  - メッセージの処理は一つずつ
  - Actor内の状態更新には排他制御が不要
    - メッセージキューの時点で逐次化されているため



提案Actor機構

- Juliaのリモートチャネル機構を利用
- リモートチャネル経由でメッセージを配送
- メッセージハンドラ関数をリモートノードで起動



Actorの記述方法の比較

- Juliaのマクロ機能を用いてハンドラ関数の記述を容易に

```
mutable struct Counter <: Actor
  v::Int64
end

@remote function add(c::Counter, v::Int64)
  c.v += v
end

@remote function sub(c::Counter, v::Int64)
  c.v -= v
end
```

```
counter = @startat 2 Counter(0)
f = add(counter, 10)
fetch(f)
```

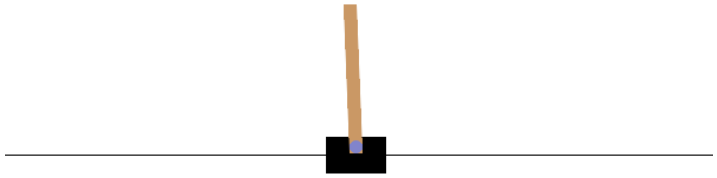
```
import ray
ray.init()

@ray.remote
class Counter(object):
    def __init__(self):
        self.n = 0
    def add(self, v):
        self.n += v
        return self.n
    def sub(self, v):
        self.n -= v
        return self.n

c = Counter.remote()

future = c.add.remote(10)

print(ray.get(future))
```

Rayによる記述

OpenAI Gym (Farama Gymnasium)

- さまざまな強化学習の課題を統一的なインターフェイスで提供
  - 2022年10月にOpenAIからFaramaに移管
- Env
  - step, reset, render, close
  - action_space, observation_space, reward_range,…

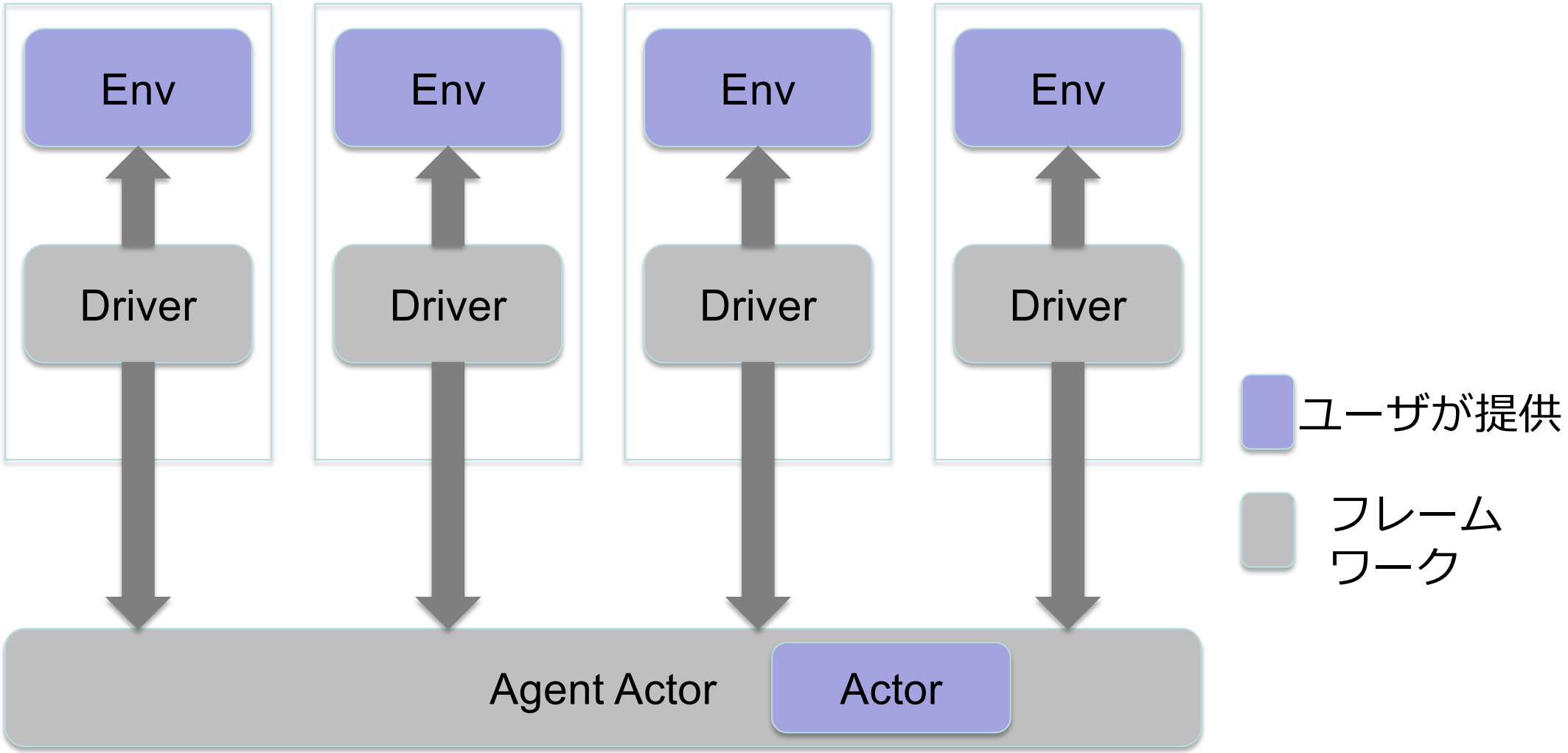
- Cart Pole (倒立振子)
  - カートに対して回転するように固定された棒を倒さないようにカートを左右に動かす
  - Observation は 位置、速度、角度、角速度
  - Actionは右に行くか左に行くか



分散強化学習フレームワークの設計

- 要請
  - 複数の「環境」を用いて1つのエージェントを訓練する
    - 「環境」のシミュレーションが重いことを想定
  - さまざまな強化学習アルゴリズム、評価環境をプラグイン可能
  - ユーザには分散環境を意識させない
    - 通常の構造体と関数として記述させる
    - それを分散環境上にフレームワークが展開させる

フレームワークの概要


Env
Env
Env
Env
Driver
Driver
Driver
Driver
Agent Actor
Actor
ユーザが提供
フレームワーク

Agentのインターフェイス

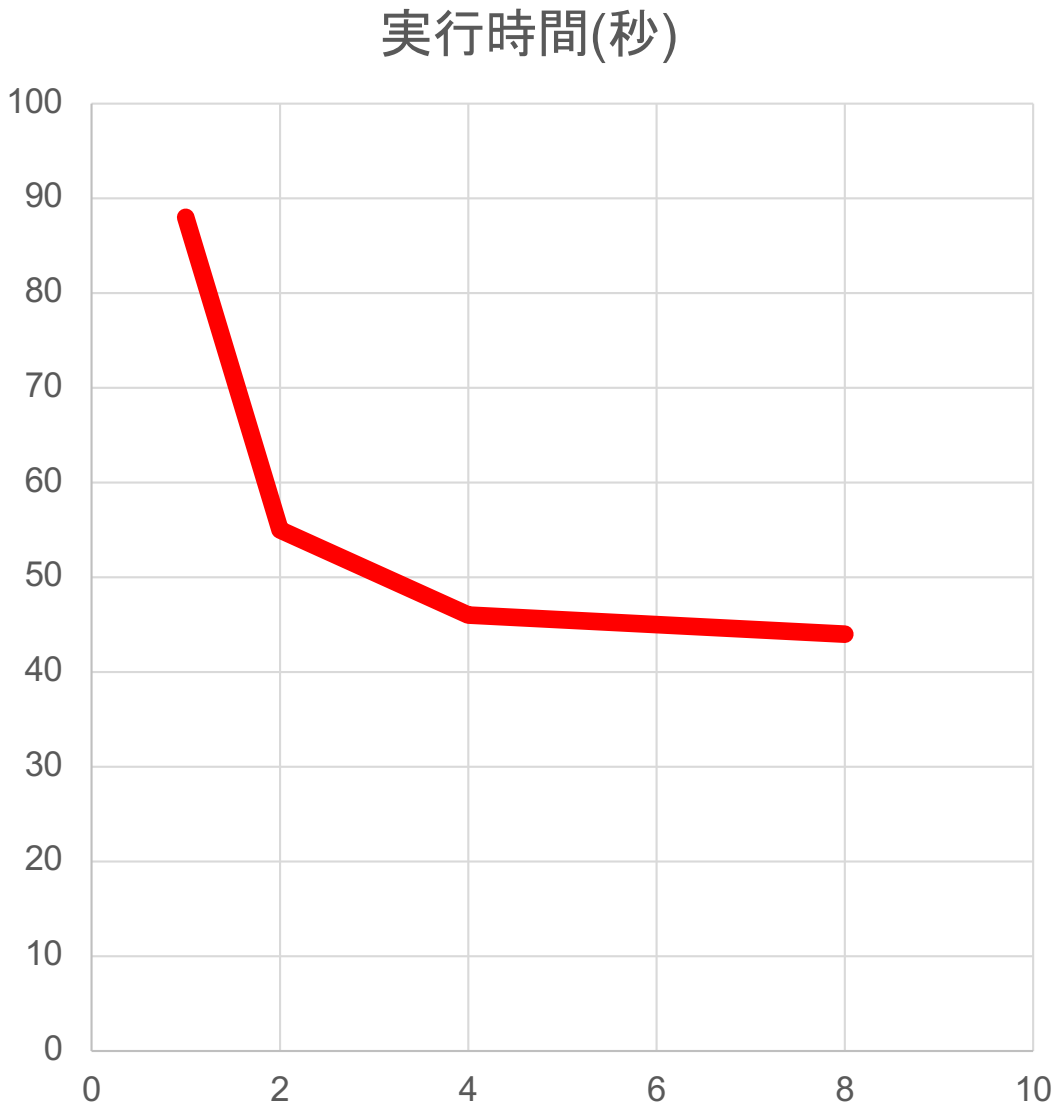
- get_action
  - 次のactionを決定
- update
  - 内部状態を報酬に応じて更新

```
mutable struct Agent
    …
end

function get_action(
    agent::Agent,
    observation::Vector{Float32},
    step::Int)
    …
end

function update(
    agent::Agent,
    observation::Vector{Float32},
    action::Int,
    reward::Float32
    observation_next::Vector{Float32})
  …
end
```

実験結果

- OpenAI Gymの cart pole
  - 環境としては軽量
- M2 Macbook Proで評価
- 基本的なQ学習

- 速度の向上は見られたが、十分ではない
  - 環境が軽いため、並列実行のメリットがない
  - オーバヘッドが大きい
- C.f. シングルプロセスでは8秒



議論

- 現状の設計ではフレームワークとエージェントの分離が完全ではない
  - 報酬を決定するロジックがフレームワーク側にある
  - エージェントのインターフェイスは要改善
- Interface(Java)もしくはTrait(Rust)のような機構がないと、安全なフレームワークの構築は困難
  - Juliaには abstract type があるが、具象型での実装を強制できない
- マクロを使用することで、ファクトリ関数が不要

おわりに

- まとめ
  - 分散強化学習フレームワークのプロトタイプを設計した
- 今後の課題
  - さまざまな環境、強化学習アルゴリズムへの対応を通じて、インターフェイスを改善
- 参考文献
  - Ray: https://www.ray.io/
  - Gymnasium: https://gymnasium.farama.org/