Raku(Perl6) のサーバーを使った高速実行
Running Raku using Raku server

165727F 氏名: 福田 光希 指導教員: 河野 真治

2019/10

1 スクリプト言語の高速実行

スクリプト言語 Raku は MoarVM という VM 上で動作するが, 起動時間が Perl5 や Python,Ruby などの主要なスクリプト言語に比べて非常に低速である. この問題を解決するために,Raku プログラムの, サーバーを用いた実行手法の提案を行う．ここでいうサーバーとは転送したスクリプトを実行する環境のことである.

またサーバーでは, サーバーに投げられた Raku をコンパイラで実行する際に, そのスクリプトが次に実行するスクリプトに影響を与えないことを保証する必要がある．この問題を解決するために, サーバーのコンテナ化を行う.

2 Raku

Raku は元は Perl5 の後継言語の Perl6 として開発されていたが, 現在は名称が変更され Raku となっている．Raku の現在の主流な実装は Rakudo である．Rakudo は MoarVM, と NQP と呼ばれる Raku のサブセット,NQP と Raku 自身で記述された Raku という構成である．MoarVM は NQP と Byte Code を解釈する.

NQP とは Not Quite Perl の略で Raku のサブセットである. その為基本的な文 法などは Raku に準拠しているが, 変数を束縛で宣言する などの違いが見られる.

この NQP で記述された Raku の事を Rakudo と呼ぶ. Rakudo は MoarVM の他に JVM,Javascript を動作環境として選択可能である.

Raku の起動は,MoarVM を起動,nqp をロード,Rakudo をロードもしくはコンパイルし, その後 JIT しながら実行する.

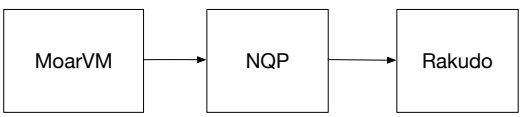

図 1: Raku の構成

3 MoarVM

MoarVM は Raku に特化した VM である.C 言語で実装されている．JIT コンパイルなどが現在導入されているが, 起動時間などが低速である問題がある． MoarVM 独自の ByteCode があり, NQP からこれを出力する機能などが存在している.

4 NQP

NQP とは Raku のサブセットである．その為基本的な文法などは Raku に準拠しているが，変数を束縛で宣言するなどの違いが見られる.

NQP は最終的には NQP 自身でブートストラップする言語であるが, ビルドの最初にはすでに書かれた MoarvM-ByteCode を必要とする．この MoarVMByteCode の状態を Stage0 と言う．Raku の一部は NQP を拡張したもので書かれている為,Rakudo を動作させる為には MoarVM などの VM,VM に対応させる様にビルドした NQP がそれぞれ必要となる．NQP は与えられた Stage0 を使い Stage1 をビルドし，その Stage1 を利用し Stage2 をビルドする事で生成できる.

5 なぜ Raku は遅いのか

通常 Ruby のようなスクリプト言語ではまず rubyVM が起動し, その後スクリプトを Byte code に変換して実行という手順を踏む. Rakudo はインタプリタの起動時間及び、全体的な処理時間が他のスクリプト言語と比較して非常に低速である。これは Rakudo 自体が Raku で書かれているため,MoarVM を起動し,Rakudo と NQP をコンパイルし, その後スクリプトの Byte code 変換というような手順で進むためである。また Raku は実行時の情報が必要であり, メソッドを実行する際に invoke が走ることも遅い原因である.

6 Abyss サーバー

ここでは Abyss サーバーについて説明する.Abyss サーバーは Raku で書かれている．クライアント側から投げら

れた Raku を実行するためのサーバーである．図 1 は Abyss サーバーを用いたスクリプト言語実行手順である．Abyss サーバーはユーザーが Raku を実行する際, クライアント側から転送されてきたファイルを事前に起動してあるサーバー側が処理し, その実行結果を返す構造となっている．

この手法を用いることで, サーバー上で事前に Rakudo を起動した Rakudo を再利用し, 投げられた Raku スクリプトの実行を行うため Rakudo の起動時間を短縮できると推測できる．

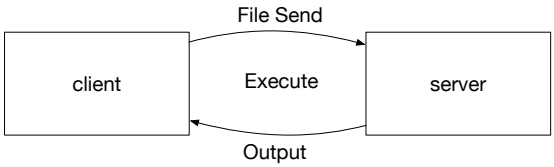

図 2: Abyss サーバーを用いたスクリプト言語実行手順

Code1 は Abyss サーバーのソースコードである．Abyss サーバーは起動すると, まず自身にファイルを転送するためのソケットを生成し, その後ファイルを受け取るための待機ループに入る．

ファイルパスを受け取るとファイルパスをバッファーに変換し読み込む, その後読み込んだバッファーを文字列にデコードし, ファイルパスの文字列を読み込み, ファイルの中身を式として評価する EVALFILE を用いて, プログラムを実行する．Code1 の 2 行目にある MONKEY − SEE − NO − EVAL は Raku 上で EVALFILE を使用可能にする pragma である．現状の Raku の EVALFILE では, 出力がサーバー側に返っているので, クライアント側から出力を見るためにクライアント側に返す必要がある．

Code 1: Abyss サーバーの source code

```
unit class Abyss::Server:ver<0.0.1>;
use MONKEY−SEE−NO−EVAL;

method readeval {
  my $listen = IO::Socket::INET.new( :listen,
                                     :localhost<localhost>,
                                     :localport(3333) );
  loop {
      my $conn = $listen.accept;
          while my $buf = $conn.read(1024) {
              EVALFILE $buf.decode;
          }
      $conn.close;
  }
}
```

Code 2: クライアント側の source code

```
my $conn = IO::Socket::INET.new( :host<localhost>,
                                 :port(3333) );

$conn.print: 'FILEPASS';
```

7 まとめ

中間予稿までに Perl6 スクリプトを投げて実行するサーバーの実装、および「自分でプロセス立ち上げて Perl6 実行する手法」と「既にあるサーバーに投げて Perl6 スクリプトを実行する手法」の差を測るために時間の計測を行った。

今回実装したサーバーでは, 別のスクリプトを実行する前にサーバーのコンテナ化をできていないので次回以降の課題とする．今回の実装では TCP ソケットを用いたが TCP ソケットを用いるとサーバーを立ち上げた際に外部からファイルを転送される可能性があるので,Unix domain socket の実装を行い, それを用いたクライアント・サーバーを作成することで安全性が高まると考えた Raku には現状 Unix domain ソケットの実装がないので、Unix domain ソケットを実装し、自分以外が実行できないようにすることが今後の課題に挙げられる．また今回例題として用いたものはスクリプト言語 Raku であったが, その他のスクリプト言語にも応用が利くかどうか検討する必要はある

今回用いた Raku の EVALFILE 自体にクライアント側に出力を返す実装追加することも今後の課題に挙げられる．

参考文献

[1] Andrew Shitov. Perl6 Deep Dive

[2] 清水隆博, 河野真治. CbC を用いた Perl6 処理系. 琉球大学工学部情報工学科平成 30 年度学位論文 (学士), 2018.

[3] Perl6 Documentation
https://docs.perl6.org (2019/10/22 アクセス)