Rakuのサーバーを用いた実行
Running with Raku server

学籍番号 175748C 氏名 大蔵 海斗　　指導教員：河野 真治

要旨

現在開発の進んでいる言語にスクリプト言語の Raku がある. Raku は起動時間が 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. Raku は, コンパイラが Raku そのもので書かれているため, 毎回コンパイラのロードとコンパイル, JIT コンパイルを繰り返すことになる. 最近のスクリプト言語では, コンパイラが自身で書かれているケースが多い. 例えば, Pypy, golang, Haskell などである. そこで, この問題を解決するために, 既にコンパイラをロードしてあるサーバーを用意し, サーバー上でスクリプト言語を実行する手法を提案している. Raku に対しては, 当研究室にて Abyss サーバーを開発している.

There is a scripting language, Raku, that is currently under development. Raku's startup time is very slow compared to other major scripting languages such as Perl5, Python, and Ruby. Raku's compiler is written in Raku itself, so loading and compiling the compiler and JIT compiling are repeated every time. In recent scripting languages, there are many cases where the compilers are written by themselves. For example, Pypy, golang, Haskell and so on. So, to solve this problem, we prepare the server which has already loaded the compilers, We have proposed a method of executing the script language on the server. For Raku, we are developing the Abyss server in our laboratory.

1　プログラミング言語 Raku

Raku は元は Perl5 の後継言語の Perl6 として開発されていたが, 言語仕様及び処理実装が Perl5 と大幅に異なっており, 言語的な互換性が存在しない. 従って現在では Raku と Perl5 は別言語としての開発方針になっている. Raku という名称は, 現在有力な処理系である Rakudo が由来となっている.

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.

Rakudo とは Raku の現在の主流な実装である. (Raku は言語名, Rakudo はコンパイラ) Raku は仕様と実装が明確に区分されており, Rakudo という実装, roast という Raku の仕様 (テストスイートがある).

Rakudo の構成は, MoarVM と呼ばれる Rakudo のために構築された VM, 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 である. Rakudo は MoarVM の他に JVM や JavaScript を動作環境として選択可能である.

MoarVM は Rakudo, NQP のために構築された VM であり, 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, 起動時間が低速であるなどの問題がある. 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.

MoarVM は NQP と ByteCode 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 における NQP は現在 MoarVM, JVM 上で動作する. 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 のバイトコードを必要とする. 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.

通常, Ruby のようなスクリプト言語ではまず VM が起動し, その後スクリプトをバイトコードに変換して実行という手順を踏む.

対して Raku は, コンパイラの Rakudo 自体が Raku と NQP で書かれているため, MoarVM を起動し, Rakudo と NQP のバイトコードを読み取り, Rakudo を起動し, その後スクリプトをバイトコードに変換して実行という手順を踏む.

そのため, 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.

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. invoke は MoarVM の method 呼び出しのバイトコードである.

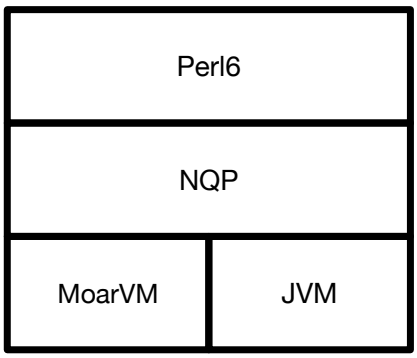

Figure 1: Rakudo の構成

2　Abyss サーバー

本研究で提案している Abyss サーバーはクライアント側から投げられた Raku スクリプトを実行するためのサーバーである.

Figure2 は Abyss サーバーを用いた Raku の実行手順である. Abyss サーバーはユーザーが Raku を直接立ち上げるのではなく, まず同一ホスト内で Abyss サーバーを起動し, ユーザーは Abyss サーバーにファイルパスをソケット通信で送り, Abyss サーバーがファイルを開き実行し, その実行結果をユーザーに返す.

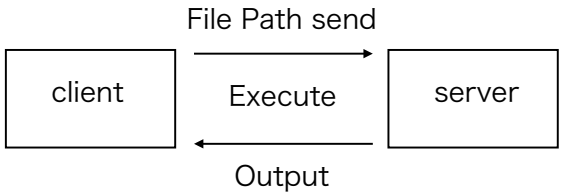

Figure 2: Abyss サーバーを用いた Raku の実行

3　Abyss サーバーの課題

Raku は開発中のため, 主なアプリケーションはコンパイラとテストコードしかない. 最初の目標として, テストの高速化を行う. 現状では.. 秒かかる. テストは.. 項目あり, 連続的に実行される. 我々の方法では, テストスクリプトをサーバーに連続的に送り実行することになる. この時に, 毎回コンパイラをロードしなおすのでは高速化にならない. サーバーからの出力はソケット経由で行われるが, 各テスト項目ごとにソケットの open/close を繰り返すのは望ましくない.

一方で, 前のテストの結果が次のテストに影響することは望ましくない. 例えば, 大域変数が残っていたりする場合である. これらの後始末をする必要がある.

複数のテストの実行や入出力をパイプライン的に実行することにより, 高速化が可能だと思われる.

パイプライン実行は並列実行と異なり, 個々のテストは並行に実行されない. サーバー上での並列実行を可能にするには, 複数の実行ワーカーを用意すればよい.

これらの内容により, テストを高速で実行することが最初の目標となる.

実際のスクリプト環境として使用する場合には, 実行するスクリプト間での干渉を抑える必要がある. しかし, JIT コンパイルされた結果を再利用しないとすると, 実行時間は不利になる. これらを調整する仕組みと API を開発し, 評価する必要がある.

現在は, Abyss サーバーの実行の再現を行なっている.

References

[1] 福田光希, 河野真治. Raku のサーバーを使った実行. 琉球大学工学部情報工学科令和元年度卒業論文.

[2] 福田光希, 河野真治. Perl6 のサーバーを使った実行. プログラミングシンポジウム論文.

[3] Raku 入門
https://raku.guide/ja/(2020/09/09 アクセス)