分散Ruby実行環境の実装

後藤翔　　千葉雄司　　土居範久

中央大学理工学研究科情報工学専攻

要約

Rubyの標準分散実行環境dRubyを改良し，新たなRuby向け分散実行環境dRuby+を開発した．dRuby+の開発目標は二つあり，一つは分散環境が提供すべき機能である分散ごみ集めとネームサービスを実現することで，もう一つは分散アプリケーションの開発を容易にする返戻値の転送機能を実現することにある．従来の分散アプリケーション開発では遠隔メソッド呼出しの返戻値を遠隔参照渡しするか，あるいは値渡しするかを，開発者が判断してソースコード上に明記する必要があったが，提案技法はこの判断が不要にすることで，開発を容易にする．

1. はじめに

Rubyには，標準で提供される分散実行環境dRuby[1]があるが，現状のdRubyには，一般的な分散実行環境が提供するサービスのうち，ネームサービスと分散ごみ集めが実装されていない．そこで我々は，dRubyをベースとした新たなRuby向け分散実行環境dRuby+を開発した．dRuby+は，分散ごみ集めとネームサービスを提供するほかに，分散アプリケーションの開発を容易にする機能を新規提供する．この機能を使うと，遠隔メソッド呼出し(RMI：Remote Method Invocation)の返戻値を値渡しすべきか，あるいは遠隔参照渡しすべきかを，開発者が考えてソースコード上に明記する必要がなくなるので，その分だけ分散アプリケーションの開発が容易になる．我々はこの機能を実現するために，返戻値を引き渡すプロトコルを改善した．本論文では，このプロトコルの詳細を示す．まず，2章で従来のプロトコルを概観し，3章では我々が提案するプロトコルを示す．4章では提案するプロトコルが遠隔メソッド呼出しの実行速度に及ぼす影響の評価結果を示す．5章は結論である．

Implementation and Evaluation of Distributed Execution Environments for Ruby

† Sho Goto, Yuji Chiba, Norihisa Doi

‡ Information and System Eng. Course, Graduate School of Science and Eng., Chuo University

2. 転送プロトコル

本論文での転送プロトコルとは，遠隔メソッド呼出し時における返戻値の渡し方を決定する手順を指す．本章では，既存の分散実行環境における転送プロトコルについて概観する．

2.1 dRuby

Rubyでは，メソッド呼出しの引数や返戻値，例外の全てを参照渡しするが，dRubyでは，RMIの返戻値を値渡しか遠隔参照渡しのどちらかで渡す．どちらで渡すのかは開発者が明示的に指定することも出来るが，dRubyの処理系に任せることも出来る．しかし，dRubyの処理系に任せると，サーバはクライアントがデシリアライズを行うために必要なクラスを持つか否かを考慮せずに，オブジェクトを値渡しする．結果として，デシリアライズに失敗することがあるが，それが問題となる場合には，開発者が遠隔参照渡しをソースコード上に明示的に指定することで解決しなくてはならない．

2.2 その他の分散実行環境

dRubyと，その他の分散実行環境における転送プロトコルの違いを表1に示す．CORBAはOMG[2]が定めた，異機種分散環境上での分散オブジェクト技術の仕様である．JavaRMIとPyroは，それぞれJavaとPythonにおける分散実行環境である．それぞれの環境において，開発者は必要に応じて，返戻値の渡し方を明示的に指定する必要がある．

	遠隔参照渡し	値渡し
dRuby	シリアライズ不可なオブジェクト	シリアライズ可能なオブジェクト
CORBA	・CORBA基本型 ・IDL定義型	valuetype型
Java RMI	UnicastRemoteObject型	・プリミティブ型 ・Serializableを実装したクラス
Pyro	Dynamic proxy利用時	Attribute Proxy利用時

表1　分散実行環境による転送プロトコルの違い

3. 実装

3.1 改良した転送プロトコル

dRuby+では開発者の負担を減らすため，転送プロトコルを改良し，クライアントがデシリアライズに失敗した場合には，遠隔参照渡しでの RMI を自動でおこなう（詳細を図 1 に示す）．これにより，クライアントがシリアライズに必要なクラスを持つか否かに関わらず，サーバ，およびクライアントの開発者は返戻値の渡し方を明示的に指定する必要がなくなる．

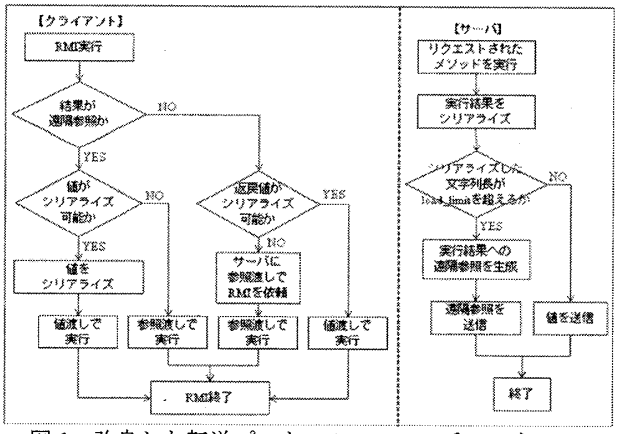

図 1　改良した転送プロトコルのフローチャート

4. 評価

本論文で提案したプロトコルには，分散アプリケーションの開発を容易にするという利点がある一方で，クライアントがクラスを持たない場合は，遠隔参照渡しするよう再リクエストをするために余分なオーバヘッドがかかるという欠点がある．このオーバヘッドが大きすぎると提案プロトコルは実用的とはいえない．そこで，このオーバヘッドの大きさを測定することを通じて，提案したプロトコルの実用性を評価した．評価に用いた PC の CPU は Opteron 1.1GHz，メモリは 2GB，OS は CentOS5.3(Kernel2.6.18)であり，RubyVM の version は 1.9.1p376 である．

評価方法は次に示す通りである．まず，サーバ側で図 2 に示すプログラムのメソッド ret を公開し，次に，クライアントからメソッド ret を遠隔呼出しし，返戻値のインスタンス変数 ins_var を参照するまでにかかる時間を計測した．計測した時間は主に分散実行環境内で消費されているので，その長短は分散実行環境の性能を表す．

```
class Test
    def initialize(num)  @ins_var = num   end
    def ret(num)         Test.new(num)    end
end
```

図 2 評価プログラム

計測は，クライアントが Test クラスを (i) 持つ (ii) 持たない双方の場合についてそれぞれ実施した．dRuby+では，(i)の場合，クライアント側で Test クラスのインスタンスをデシリアライズ可能なため，インスタンス変数の取得はクライアントのインスタンスに対して実行されるが，(ii)の場合は遠隔参照経由でサーバのインスタンスに対して実行される．また比較のため，dRuby で値渡しと遠隔参照渡しをそれぞれ指定した場合の実行時間も計測した．計測結果を表 2 に示す．

	回数	dRuby [秒]		dRuby+ [秒]
		値渡し	遠隔参照渡し	指定なし
(i)	100	0.049586	0.116171	0.051100
	1000	0.461348	実行不可	0.530794
	10000	4.618149	実行不可	5.415025
(ii)	100	実行不可	0.122307	0.172179
	1000	実行不可	実行不可	1.659996
	10000	実行不可	実行不可	17.904589

表 2　計測結果（Test クラスを(i)持つ (ii)持たない）

計測の結果，dRuby+は，(i) クライアントが Test クラスを持つとき，dRuby+内部の判定ルーチン（3.1 節参照）により，dRuby の値渡しでの実行時間と比べて 3～17%遅くなる．(ii) Test クラスを持たないときは，dRuby では参照渡しを明示的に指定しないとクライアントがデシリアライズに失敗するため実行できないが，dRuby+では指定の必要なく実行可能になった．また，dRuby で参照渡しを指定した場合と比べると，dRuby+では 41%遅くなった．これは，3.1 節で述べた再リクエストと分散ごみ集めにかかるコストが原因と思われる．(i)(ii)どちらの場合でも，dRuby では遠隔参照渡しによる 1000 回以上の実行ができなかった．この原因はサーバ側でごみ集めが起き，遠隔参照されているオブジェクトを誤回収した事にある．dRuby+では分散ごみ集めを実装しているため，問題なく実行できた．

5. まとめ

本研究では，実用的な分散アプリケーション開発が可能となる分散実行環境を目指し，ネームサービスと分散ごみ集め，改良した転送プロトコルを提供する dRuby+を開発した．評価の結果，分散ごみ集めと転送プロトコルによるオーバヘッドにより 3～41%実行時間が遅くなるものの，開発者が返戻値の渡し方を明示的に指定する必要なく，RMI が可能となることが分かった．

参考文献

[1] 関将俊，「dRuby による分散・Web プログラミング」，オーム社，2005
[2] Object Management Group, http://www.omg.org/