動的コード生成を用いた適応的移動コードに関する研究

川崎 大輔

北陸先端科学技術大学院大学 情報科学研究科

1999年2月15日

キーワード： 動的コード生成, 移動コード, テンプレート, Java, 適応.

Telescriptや Java等に見られるように、移動コードはネットワーク上のアプリケーションに新しいアプローチをもたらした。これまでのネットワークアプリケーションは、リモートプロシージャコール (RPC) を主に用いている。RPCパラダイムは1970年代に発明され、コンピュータ間の通信を、一方のコンピュータから他方のコンピュータのプロシージャを呼び出すようなものだと考えた。ネットワークを通じて運ばれるメッセージは、プロシージャへのリクエスト、もしくはプロシージャの実行結果のレスポンスである。リクエストにはプロシージャと、引数としてのデータが含まれる。レスポンスには実行結果のデータが含まれており、プロシージャそのものはそれを実行するコンピュータ側に有った。リモートプロシージャコールでは、1回のコンピュータのインタラクションに2回の通信が必要になるのが大きな特徴である。最初にサーバに対してプロシージャ実行を依頼し、つぎにサーバから動作結果が送られてくる。この間、「リモートプロシージャコールの処理中ずっとコンピュータ同士が接続されていなければならない」のである。これに対し、リモートプログラミング (RP) では、コンピュータ間の通信において、あるコンピュータから他のコンピュータに対してプロシージャをコールするだけではなく、実行されるプロシージャそのものを転送してしまうことができる。リモートプログラミングの顕著な特徴として、ユーザーのコンピュータとサーバとの間でネットワークを介し、1度移動コードを送れば、その後ネットワークを切断しても処理が継続される。つまり「処理中にコンピュータ同士が接続されている必要がない」のである。これが従来なかなかできなかったことであり、移動コードのもっとも顕著な利点である。

移動コードは、様々な計算機環境でアプリケーションの実行を可能とした。しかし、様々な計算機環境において実行することはできるようになったが、様々な計算機環境に適応した振る舞いを行うという点について考慮されていない。移動先の環境は、ほとんどの

場合、その計算機の使用目的によって実に様々である。例えば、OS、CPU、ネットワーク環境、mpegエンコーダ等の特殊なデバイス等があげられる。この時、移動先の環境に沿った動作を行うことで、実行効率の改善や、容易なオペレーションが得られる場合がある。このような場合、環境の変化にアプリケーションが対応していくことは有効である。

しかし、環境の変化に対応できるアプリケーションを作成するのは容易ではない。様々な環境の変化に対応するためには、あらゆる環境を想定し、その環境に対応した複数のコードを書かなくてはいけない。

例えば、描画命令を多量に含む動的適用可能な移動コードを作成したいとする。そして環境によっては描画命令を高速に実行できるハードウェアが利用可能であるとする。

- 実行時にドラスティックにディスパッチを行い、描画命令を選択しながら実行するようにするという方法が考えられる。この方法は、プログラムを記述しやすいという長所がある反面、実行が遅いという欠点がある。

- ハードウェア支援を用いないメソッド全体、およびハードウェア支援を用いたメソッドを記述し、ディスパッチによってメソッドを起動し分ける方法である。この方法は、実行速度が速いという長所がある反面、冗長な記述を強いられるという短所がある。

ここで、環境の変化に着目してみる。環境の変化は、「常に変化する可能性のある環境」と「マシン固有の環境」に大別できる。環境の変化に動的に応じて、ディスパッチを行ない、現在の環境に最適なメソッドを呼び出すことで、どちらの環境の変化にも対応することはできる。しかし「マシン固有の環境」で、毎回ディスパッチを行なうことは無駄である。マシン固有の環境としては、OS、CPU、ネットワーク環境、特殊なハードウェアプロセッサといったものが上げられる。また、最近は画面サイズや、色数が動的に変更できる機構もOSに備わってきているが、実際動的に変更することは希である。つまり、移動先において適応の対象となる環境情報は、移動コードが実行環境に移動した時点で、殆ど静的に決定することができる。そこで、移動コードを実行する前に、実行環境に応じた新しいコードを生成することで、ディスパッチを行うオーバーヘッドを無くすことができる。

この事を上の例に当てはめて考えてみる。実行時に静的に決定される情報を用いて、実行環境に応じたコードを生成することにより、例で述べた両方の長所を保ちながら、欠点を取り除くことが可能である。

実行時にコードを生成する方法として良く用いられるものに動的コード生成がある。動的コード生成は、プログラムの再コンパイルに比べて大変高速に実行コードを生成することができる。動的コード生成法には次のようなものがあり、それぞれに特徴を持っている。

- 中間コードから、動的コード生成によりネイティブコードを生成する。

- 部分評価による最適化を用いたもの。

- テンプレートと、テンプレートを埋めるコード片を用いる。

本研究では、静的に決定される情報を元に、実行環境に適応したコードを、実行時に生成することで、記述性を確保しながら、処理速度の向上を得るための手法の提案を行った。また、Javaを用いて効率の良い等価な命令への置き換え、および不要コードの除去を行える処理系を、プロトタイプとして作成した。このプロトタイプは、Javaのコンパイラや、VMや、JIT等にはまったく手を加えていないのでJavaの利点を損なうようなことはない。動的コード生成では、定数伝播、ループ展開、部分評価による不要コードの除去といった最適化の手法が取られている。今回作成したプロトタイプでは、動的コード生成で行う高度な最適化をほとんど行っていないにもかかわらず実行速度の向上が得られた。つまり、今回作成したプロトタイプに動的コード生成に用いられている高度な最適化を施すことで、更なる実行速度の向上が期待できる。