RM-010

キャッシュによるモバイルエージェントの移動高速化
Speeding Up Mobile Agent Migration using Caching

東野 正幸 † 本村 真一 ‡
Masayuki HIGASHINO Shinichi MOTOMURA
笹間 俊彦 † 川村 尚生 † 菅原 一孔 †
Toshihiko SASAMA Takao KAWAMURA Kazunori SUGAHARA

1 はじめに

エージェントとは，人間の代理となって自律的にタスクの処理を行う，エージェント実行環境上で動作するプログラムであり，モバイルエージェントとは，ネットワークに接続されたエージェント実行環境間を移動可能なエージェントである．モバイルエージェントは，タスクが記述された複数のプログラムコードを逐次的に読み込み，解釈しながら動作する．また，自律的な判断に基づいて，タスクが記述されたプログラムコードを追加・削除することで，未知の環境へ柔軟に適応する能力を持たせることができる．そのようなモバイルエージェントが，エージェント実行環境間を移動しながらタスクの処理を継続するには，モバイルエージェントが移動する際に，エージェントと共にプログラムコードも転送する必要がある．なぜなら，前述したようにモバイルエージェントはタスクとなるプログラムコードを動的に追加・削除するため，それぞれのエージェント実行環境に固定的に配置しておくことはできないからである．

モバイルエージェント技術をベースとして分散アプリケーションを開発する場合，その実現に必要な機能はモバイルエージェントとして実装される．実用的で多機能な分散アプリケーションを開発しようとする場合，モバイルエージェントが処理するタスクは複雑化・肥大化するため，モバイルエージェントの移動時に転送しなければならないプログラムコードの量も増大する．これは，モバイルエージェントの移動にかかるオーバーヘッドとなり，分散アプリケーション全体のパフォーマンス低下に繋がる．そのため，モバイルエージェントの移動を高速化することは非常に重要な課題である．

そこで本稿では，モバイルエージェントが移動する際に，モバイルエージェントが処理するタスクが記述されたプログラムコードを，移動元と移動先のエージェント実行環境にキャッシュする方式を提案する．提案方式によってプログラムコードの転送量を削減することで，モバイルエージェントの移動を高速化することが可能となる．また，提案方式の実装を行い，実際的な分散アプリケーションに適用することにより，その有効性について評価を行う．

以降，本稿では特に断りが無い限り，モバイルエージェントを単にエージェントとよぶこととする．また，モバイルエージェントが行うタスクが記述されたプログラムコードを単にプログラムコードとよぶこととする．

†鳥取大学大学院 工学研究科 情報エレクトロニクス専攻
‡鳥取大学 総合メディア基盤センター

2 従来方式の課題

従来から提案されているプログラムコードの転送方式は，主に以下で述べる 3 種類に分類できる [1]．この章では，それぞれの概要とその課題について述べる．

2.1 固定配置方式

エージェント実行環境にプログラムコードをあらかじめ固定的に配置しておく方式である．エージェントの移動時にはプログラムコードの転送を行わず，エージェントはエージェント実行環境に既に配置されているプログラムコードのみを利用してタスクの処理を行うため，他の方式よりもエージェントが高速に移動可能である．

しかし，この方式ではプログラムコードがあらかじめ配置されていないエージェント実行環境へエージェントを移動させて所望のタスクを処理させることができない．そのため，ネットワークに新たなエージェント実行環境を追加するだけでエージェントが自己組織的にアプリケーションをスケールアウトするようなシステムを構築することは困難となる．また，アプリケーションに新たな種類のエージェントを追加する際には，全てのエージェント実行環境にそのエージェントのためのプログラムコードを配置するという多大な手間を要する．

2.2 オンデマンド転送方式

エージェントの移動後に，移動先で実際にプログラムコードが必要になった時点で，プログラムコードを移動元から移動先へ転送する方式である．この方式は，プログラムコードを必要な量だけ転送するため，トラフィックを最小に抑えることが可能である．しかし，いつ移動先でプログラムコードが必要になるかは解らないため，エージェント実行環境をネットワークに常時接続しておく必要があり，ネットワークにエージェント実行環境を任意のタイミングで追加・削除するようなアプリケーションを構築することが困難となる．

2.3 一括転送方式

モバイルエージェントが移動する際に，エージェントがタスクの処理に必要とする全てのプログラムコードを一括して転送する方式である．この方式は，1 度だけの通信でエージェントの移動を完結することができるため，エージェント実行環境が常時接続されていないネットワーク環境で優位である．しかし，1 章で述べたように，プログラムコードの転送が大量になり，それによるエージェントの移動にかかるオーバーヘッドがアプリケーション全体のパフォーマンスに影響するため，特にイン

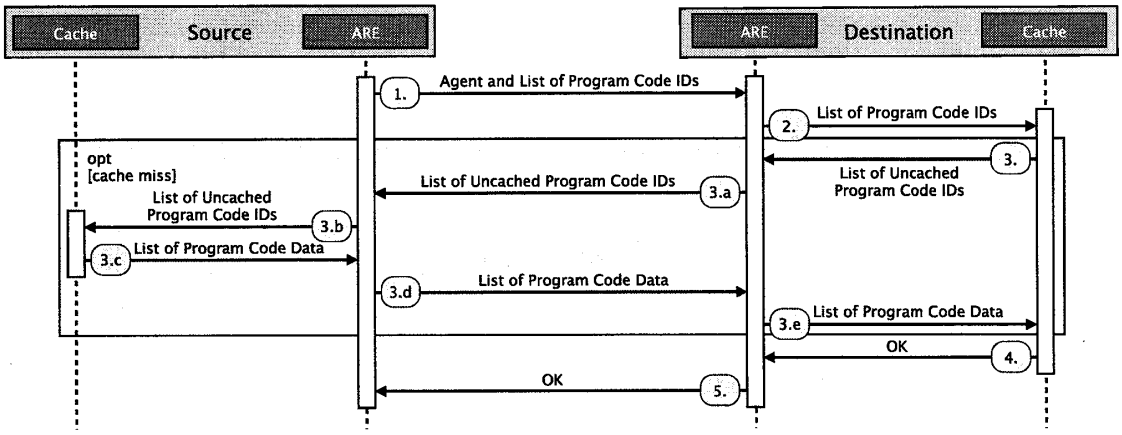


図 1: エージェントの移動における通信シーケンス．ARE(Agent Runtime Environment) はエージェント実行環境，Cache はエージェント実行環境のキャッシュ装置，Source は移動元，Destination は移動先であることを表す．opt[cache miss] で囲まれたシーケンスはキャッシュミスが発生した場合にのみ実行される．

ターネットなどの通信速度が遅い環境でのパフォーマンス改善が困難となる．

3 提案方式の設計

我々は，従来方式の課題を全て解決するプログラムコードの転送方式として，エージェントがプログラムコードの配置されていないエージェント実行環境へ移動しても継続して活動可能で（2.1 節），エージェント実行環境がネットワークに常時接続されていなくてもかまわず（2.2 節），そしてエージェントが高速に移動可能な（2.3 節），プログラムコードの転送方式を提案する．提案方式では，2.3 節で述べた一括転送方式にキャッシングを組み合わせることでエージェントの移動の高速化をはかる．そのため，エージェントの移動に関する通信方式の拡張と，エージェント実行環境へのプログラムコードのキャッシュ装置の構築を行う必要がある．また，キャッシュ装置によるプログラムコード管理のために，これらのプログラムコードに対してユニークな識別 ID を必ず付与しなければならない．以降，この章ではそれらの設計について述べる．

3.1 プログラムコード ID

プログラムコード ID はプログラムコードをユニークに識別するための ID である．キャッシュ装置がプログラムコードを管理する目的で使用するため，必ずプログラムコードに付与しておかなければならない．また，プログラムコード ID を意図的に重複させることでキャッシュを汚染可能であることを考慮すれば，これは名前空間のようにアプリケーションの開発者が工夫して設定するようなものではなく，エージェント実行環境が自動的に生成して固有性を保証するものでなければならない．

3.2 エージェントの移動

提案方式では，エージェント実行環境にプログラムコードをキャッシュするためのキャッシュ装置を設置する．エージェントが移動する際に，移動先で必要とするプログラムコードが移動先のキャッシュに全て存在していた場合をキャッシュヒットとよび，逆に，不足しているために移動元から移動先へプログラムコードを転送する必要が有る場合をキャッシュミスとよぶこととする．

図 1 にエージェントの移動における通信シーケンスを示す．処理手順は次のとおりである．

1. 移動元は，移動先へエージェントとプログラムコード ID のリストを送信する．プログラムコード ID は，そのエージェントがタスクの処理に必要とするプログラムコードに対応するものである．
2. 移動先は，受信したプログラムコード ID のリストを元に，キャッシュ装置にプログラムコードが格納されているか調べる．
3. もしキャッシュされていないプログラムコードがある場合は，キャッシュミスとなり，以下の処理を行う．
   (a) 移動先は，キャッシュされていないプログラムコードのプログラムコード ID のリストを，移動元へ送信する．
   (b) 移動元は，受信したプログラムコード ID のリストを元に，キャッシュ装置へ問い合わせる．
   (c) 移動元は，キャッシュ装置からプログラムコードを取得する．
   (d) 移動元は，取得したプログラムコードを，移動先へ送信する．
   (e) 移動先は，受信したプログラムコードをキャッシュへ格納する．
4. エージェント実行環境間のプログラムコード転送処理を完了する．
5. エージェントの移動処理を完了する．

表 1: エージェントの移動に要する時間に関する実験で用いた計算機の構成

OS	Mac OS X 10.6.2
CPU	2.66 GHz Intel Core 2 Duo
Memory	4 GB 1067 MHz DDR3
Network	10/100/1000BASE-T Ethernet
UTP Cable	Category 5E
JRE Version	1.6.0_17

表 2: 実際的なアプリケーションにおける評価で用いた計算機の構成

OS	Turbolinux 10 (Kernel 2.6.0)
CPU	3.0 GHz Intel Pentium 4
Memory	1 GB
Network	1000BASE-T Ethernet
UTP Cable	Category 5E
JRE Version	1.5.0

4 実装

提案方式を，3 章に基づいて Maglog[2][3] 上に実装した．Maglog(Mobile AGent system on proLOG) とは，我々が開発しているモバイルエージェントフレームワークで，モバイルエージェントの構築環境と実行環境を提供する．Maglog は Java[4] で実装されており，エージェントは MaglogAgent クラスとして実現されている．また，MaglogAgent クラスは，Prolog インタプリタの Java 実装である PrologCafé[5] の Prolog クラスを拡張しており，エージェント自身は Prolog のインタプリタである．Maglog におけるエージェントは，Java のクラスに変換された Prolog 述語を，動的に読み込み，実行しながら活動する．Maglog においてキャッシュするプログラムコードは，Prolog 述語から Java クラスへ変換されたバイトコードとなる．プログラムコード ID は，バイトコードに対してハッシュ関数（SHA-1）を用いて生成した．

エージェントの移動と，プログラムコードの転送は，Java Object Serialization[6] によってそれぞれのオブジェクトをバイト配列へ変換し，HTTP/1.1 プロトコルを用いて転送することで実現した．

5 評価

5.1 エージェントの往復移動に要する時間

提案方式の有効性を確認するために，従来方式と提案方式について，エージェントの往復移動に要する時間に関する比較実験を行った．

実験環境 表 1 に示す計算機 2 台を Ethernet で接続した LAN を構築し，それぞれのコンピュータで Maglog のエージェント実行環境を 1 つずつ起動させた．

実験内容 実験では，固定配置方式（2.1 節），一括転送方式（2.3 節），そして提案方式（3 章）について，1 つのエージェントを 2 つのエージェント実行環境間で繰り返し往復させ，それぞれの往復に要した時間を測定した．往復させるエージェントは，タスクを処理するために 1020 個のプログラムコード（合計 842KB）が必要なものを使った．測定は，ネットワーク環境が 10, 100, 1000BASE-T の場合についてそれぞれ行った．

実験結果 実験結果のグラフを図 2 に示す．1000BASE-T（図 2(c)）のような通信が高速なネットワーク環境の場合は，提案手法の効果は僅かしかみられない．しかし，100BASE-T（図 2(b)）や，10BASE-T（図 2(a)）の場合をみると，2 回目以降の往復について，提案手法が一括転送方式よりも高速で，かつ固定配置方式に近い移動速度になっていることがわかる．特に，10BASE-T（図 2(a)）においては，提案方式は一括転送方式に比べて 10 倍近い高速化が実現できている．この傾向から，さらに低速なネットワーク環境になるほど提案方式が効果的であると推測できる．また，本実験は LAN 環境で行ったが，通信速度が遅いインターネット環境でも同様の効果が期待できる．

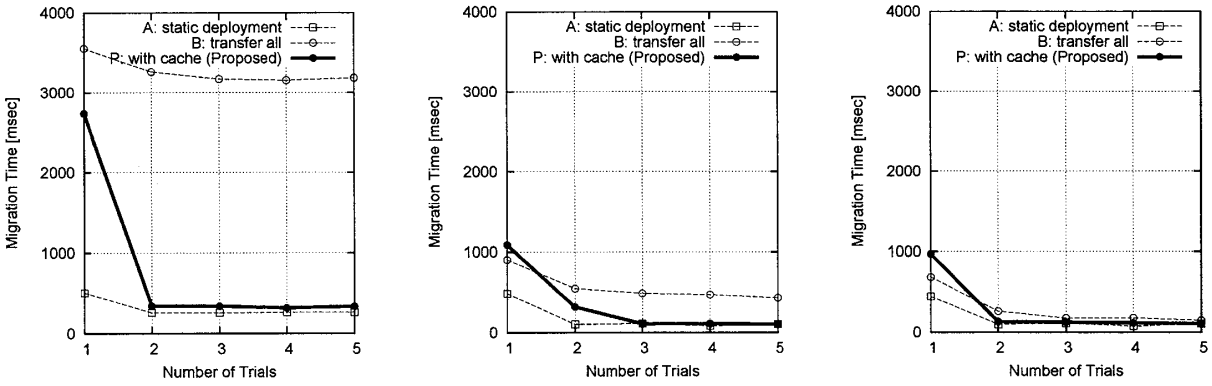
しかし，それぞれのグラフの 1 回目の往復時の移動時間からわかるように，提案手法はキャッシュを用いた高速化であるため，1 回目の往路にはついてキャッシュの効果は無く，むしろ，プログラムコード ID の転送や，キャッシュミスによる参照の空振りなどの，キャッシュ機構自体のオーバーヘッドによって，他の方式よりも遅くなる場合があることに注意が必要である．

5.2 エージェントの巡回移動に要する時間

前節の実験では，1 回目の往路では提案方式が他方式よりも遅い結果となることを述べた．そこで，提案方式が他方式よりも不利となる移動が連続する移動パターンである，巡回移動についての実験を行った．巡回移動とは，エージェントが，複数のエージェント実行環境を一度しか訪れずに渡り歩くような移動のことで，常にキャッシュが効かない移動を繰り返すことになる．

実験環境 表 1 に示す計算機 10 台を Ethernet で接続した LAN を構築し，それぞれのコンピュータで Maglog のエージェント実行環境を 1 つずつ起動させた．

実験内容 実験では，エージェントが 10 台のエージェント実行環境を巡回移動するのに要する時間を測定した．移動させるエージェントは，前節の実験と同様のものとした．ネットワーク環境は，提案方式による高速化の効果が最も薄くなる高速回線を想定して 1000BASE-T とした．



(a) 10BASE-T　(b) 100BASE-T　(c) 1000BASE-T

図 2: エージェントの往復移動の試行回数と移動時間．横軸は往復移動の試行回数，縦軸は 1 回の往復に要した時間を表す．データ系列のうち，A は初期配置方式，B は一括転送方式，P は提案方式を表す．提案方式は，特に低速な通信速度環境で有効であることがわかる．

実験結果　実験結果は，1 回目の 10 台間巡回移動について，一括転送方式は 5576 ミリ秒で完了したのに対して，提案方式は 6361 ミリ秒要した．すなわち，キャッシュ機構そのものが 10%のオーバーヘッドとなったが，1000BASE-T という高速な通信環境での 10%であることを考慮すれば，実用上は問題ないと考えられる．また，実際的なアプリケーションで，巡回移動するようなエージェントを使わざるを得ない場合は，固定配置方式を使うなど、方式の使い分けを行えば，このようなオーバーヘッドを回避することが可能である．

5.3　実際的なアプリケーションにおける評価

実際的なアプリケーションにおける提案方式の有効性を評価するために，我々が Maglog を用いて開発している会議日程調整システム [7, 8] を用いた実験を行った．我々が開発している会議日程調整システムとは，モバイルエージェントが，ネットワークに接続されたコンピュータ間を移動しながら，コンピュータのユーザに対して会議に参加可能な日程の収集や交渉を積極的に行うことで，会議の開催日時を速やかに決定するためのアプリケーションである．

このシステムでのエージェントは，会議の参加者に対する予定の収集や，ログイン認証，参加者の予定が重なった際の日程交渉，参加者への日程調整結果の通知などを行っており，システム内では多数のモバイルエージェントが頻繁に活動している．また，複数の会議日程を同時に調整する場合においては，これらのエージェント数がさらに増加し，システムのパフォーマンスが低下するという問題があった．

実験環境　表 2 に示す計算機 11 台を Ethernet で接続した LAN を構築し，それぞれの計算機で Maglog のエージェント実行環境を 1 つずつ起動させた．

実験内容　実験では，一括転送方式と提案方式について，11 台のエージェント実行環境で構築した会議日程調整システムにおいて，会議日程調整に要する時間を測定した．ただし，本来のシステムでは会議の参加者が行う作業である，予定の入力や，予定が合わない際の交渉の入力については，回答を即座に自動で行うようなプログラムを作成し，これで対応した．

実験結果　1 回目の会議日程調整に要する時間については，一括転送方式と提案方式に差は見られなかった．しかし，2 回目の会議日程調整以降は，一括転送方式は約 25 秒を要したのに対して，提案方式は約 12 秒で完了した．以上の結果より，実際的なアプリケーションでの提案方式の有効性が確認できた．

6　おわりに

本稿では，モバイルエージェントがタスクの処理に必要とするプログラムコードを，ネットワークに接続されたモバイルエージェント実行環境にキャッシュすることにより，エージェントの高速な移動を実現する方式の提案を行った．また，提案方式を我々が開発しているモバイルエージェントフレームワーク Maglog 上に実装し，現実的なアプリケーションを用いて実験を行った結果，アプリケーションのパフォーマンスを改善できることを確認した．

参考文献

[1] 佐藤一郎: モバイルエージェント技術と研究動向 (< 特集 > 情報プラットフォーム), *NII journal*, Vol. 3, pp. 53–66 (2001).

[2] Motomura, S., Kawamura, T. and Sugahara, K.: Logic-Based Mobile Agent Framework with a Concept of "Field", *IPSJ Journal*, Vol. 47, No. 4, pp. 1230–1238 (2006).

[3] Kawamura, T., Motomura, S. and Sugahara, K.: Implementation of a Logic-based Multi Agent Framework on Java Environment, *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems* (Hexmoor, H.(ed.)), pp. 486–491 (2005). Waltham, Massachusetts, USA.

[4] Oracle Corporation: Developer Resources for Java Technology, Web (2010). http://java.sun.com/.

[5] Banbara, M. and Tamura, N.: Translating a Linear Logic Programming Language into Java, *Proceedings of the ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages* (M.Carro, I.Dutra et al.(eds.)), pp. 19–39 (1999).

[6] Oracle Corporation: JDK 6 Serialization-related APIs & Developer Guides – from Sun Microsystems, Web (2010). http://java.sun.com/javase/6/docs/technotes/guides/serialization/.

[7] Kawamura, T., Motomura, S., Kagemoto, K. and Sugahara, K.: Meeting Arrangement System Based On Mobile Agent Technology, *Proceedings of the 2nd International Conference on Web Information Systems and Technologies*, pp. 117–120 (2006). Setubal, Portugal.

[8] Kawamura, T., Hamada, Y., Sugahara, K., Kagemoto, K. and Motomura, S.: Multi-Agent-based Approach for Meeting Scheduling System, *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp. 79–84 (2007). Waltham, Massachusetts, USA.