論　文

キャッシュによるエージェントの移動効率化

東野　正幸[†a)]　　高橋　健一[†b)]　　川村　尚生[†c)]　　菅原　一孔[†d)]

Effective Mobile Agent Migration Based on Code Caching

Masayuki HIGASHINO[†a)], Kenichi TAKAHASHI[†b)], Takao KAWAMURA[†c)], and Kazunori SUGAHARA[†d)]

あらまし　モバイルエージェントシステムは，モバイルエージェントと呼ばれるソフトウェアが自律的に計算機間を移動・協調しながら，特定の処理を行うシステムである．モバイルエージェントは，計算機間を移動する際に特定の処理内容が記述されたプログラムコードをもち運ぶことで，移動先の計算機で柔軟な処理を行うことができる．しかしソフトウェアの規模が大きい場合，モバイルエージェントの移動時に転送されるプログラムコードの数やサイズが大きくなるため，モバイルエージェントの移動時間が増加してシステムの性能が低下してしまう問題がある．そこで本論文では，モバイルエージェントの移動時に転送されるプログラムコードをキャッシュする手法を検討し，その有効性を評価する．

キーワード　モバイルエージェント，プラットホーム，キャッシュ

1. まえがき

近年，インターネットの世界的な普及に伴い，大規模なコンピュータネットワークを利用した多種多様なサービスが提供されている．このようなサービスを提供するシステムを実現するために，ネットワーク内に分散して存在しているソフトウェアコンポーネントを協調させて一つのシステムとして機能させる手法が提案されているが，これには複雑なプログラミングが要求される．このようなソフトウェアコンポーネント間の協調をエージェントの移動という概念を用いてモデル化することで，より人間が理解しやすいモデルでシステムを開発するための技術としてモバイルエージェントが提案されている [1], [2]．我々もこれまでに，モバイルエージェント技術を用いた会議日程調整システム [3] や分散型 e-Learning システム [4] を提案し，その有効性を示してきた．

このようなモバイルエージェント技術に基づいた分散システムでは，モバイルエージェントの移動に処理対象のデータだけでなく処理に必要なプログラムコードやプログラムの処理状態を表す実行時状態の転送を伴う．このため，モバイルエージェントの移動はソケット通信や RPC といったモバイルエージェント技術を用いない通信方式に比べて通信量が多くなり，その移動に要する通信時間が長くなることで，分散システムの性能が低下する [5]．

このため，モバイルエージェントシステムの性能向上を目的とした様々な研究 [6]～[10] が行われている．しかし，これらの研究の多くは，エージェントのデザインパターンや協調戦略と呼ばれる，特定の用途に最適化した協調アルゴリズムによって通信量や通信時間を削減する手法であるため，適用範囲が限定される．また，エージェントの移動による通信とエージェント間のメッセージ交換による通信とがトレードオフの関係となる場合がある．

そこで本論文では，エージェントの協調アルゴリズムではなく，エージェントの移動機構を改善することによってエージェントの移動時間を削減する手法を提案する．エージェントの移動機構の改善により，メッセージ交換による通信には影響を及ぼさず，エージェントの移動時に生じる通信のみを削減できる．また，エージェントの移動機構による手法と，協調アルゴリズムによる手法は，互いに独立した異なる手法であるため，これらの技術は共存させて併用することができ

† 鳥取大学大学院工学研究科情報エレクトロニクス専攻，鳥取市
Graduate School of Engineering, Tottori University, 4–101 Koyama-Minami, Tottori-shi, 680–8552 Japan
a) E-mail: s032047@ike.tottori-u.ac.jp
b) E-mail: takahashi@ike.tottori-u.ac.jp
c) E-mail: kawamura@ike.tottori-u.ac.jp
d) E-mail: sugahara@ike.tottori-u.ac.jp

る．そこで提案手法では，通信効率を高めるために，エージェントの移動機構にキャッシュを適用することを検討する．

提案手法では，エージェントの移動時にプログラムコードを移動元と移動先の計算機にキャッシュする．同じエージェントが移動してきた場合や，異なるエージェントが同じプログラムコードをもっている場合は，それらのキャッシュを利用して無駄なプログラムコードの転送を抑制する．更に，複数のエージェントが同時に移動する場合には，プログラムコードが重複して転送されないように制御する．

本論文の構成は以下のとおりである．**2.** では，エージェントを構成する内部構造を定義し，キャッシュの対象とする領域を検討する．**3.** では，キャッシュを用いたエージェントの転送プロトコルを提案する．**4.** では，実装した提案手法を評価する．**5.** では，関連研究を示すとともに，提案手法と既存手法の違いを述べる．最後に **6.** で本論文をまとめる．

2. キャッシュ構造

2.1 エージェントの内部構造

本論文では，キャッシュによりエージェントの移動時間を削減する手法を提案する．そこで，まずエージェントの内部構造を定義する．モバイルエージェントシステムの構成を図 1 に示す．エージェント実行環境では，複数のエージェントが並列に動作し，それらのエージェントはネットワークを介してエージェント実行環境間を移動する．エージェントは，実行時状態領域，アプリケーション領域，及びプログラムコード領域の 3 領域から構成される [11]．実行時状態領域は，エージェントのコールスタックやプログラムカウンタといった，処理を実行しているエージェントの状態を表す情報を保持する．アプリケーション領域は，アプリケーション固有のデータを保持する領域であり，エージェント固有の設定や処理対象のデータなどを保持する．プログラムコード領域は，エージェントの動作が記述されたプログラムコードのセットを保持する．ここで示した構成は，OMG が定めた仕様である MASIF Specification [12] や FIPA の Agent Management Specification [13], [14] で定められている仕様を包含する．

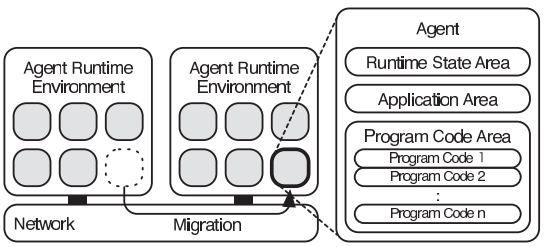


図 1　モバイルエージェントシステムの構成
Fig. 1　The structure of a mobile agent system.

2.2 キャッシュ可能領域の検討

エージェントが処理を行えば，それとともにデータの内容も変化する．常に変化するデータはキャッシュすることができない．そこで，エージェントを構成している領域の中で，変化する領域と変化しない領域を区別し，変化しない領域だけをキャッシュの対象とする．

エージェントの実行時状態領域は，コールスタックやプログラムカウンタが含まれているため，エージェントの動作とともに変化する．例えばエージェントが移動するだけでもエージェントの実行時状態領域は変化する．エージェントの実行時状態領域から変化していない部分だけを抽出してキャッシュすることも考えられるが，その実装は非常に複雑である．一般的に，コールスタックやプログラムカウンタが格納されているメモリ領域は，プログラミング言語の API から隠ぺいされており，そのようなメモリ領域を制御するためには，そのプログラミング言語の実行環境を改変する必要がある．例えば，モバイルエージェントシステムが JVM のような既存の実行環境で動作する場合には，既存の実行環境のアップデートに合わせて改変部分を対応させていく必要があり，ソフトウェアの保守に多大なコストを要する．

アプリケーション領域は，多くの場合で，特定のモバイルエージェントシステムが独自に定めたデータ構造か，エージェントの開発者が自由に定義できるデータ構造となっており，この領域のデータ構造がキャッシュ可能か否かを判別する一般的な定義を与えることは難しい．

プログラムコード領域はプログラムコードのセットが格納されている．一般的にプログラムコードは名前で識別されることから，プログラムコード領域はプログラムの名前と実体のエントリから構成される連想配列として統一的に表現できる．例えば JVM の場合，プログラムコードの名前と実体は完全修飾クラス名と Java バイトコードに対応する．

そこで本論文では，プログラムコード領域をキャッシュの対象とし，プログラムコード領域に含まれてい

る複数のプログラムコードを個別に管理することとする．

2.3 キャッシュするデータの識別

ソフトウェア開発では，プログラムコードの再利用性を高めるためにプログラムコードを部品化する方法がとられる．エージェントシステムにおいても同様に，異なるエージェントで同じプログラムコードが再利用される．例えば，エージェント開発者がエージェントをプログラムする際にライブラリを利用する場合や，コードクローン（ソースコードの一部を複製することにより作られた重複するコード）が発生した場合などでは，異なるエージェントで同じプログラムコードが利用される．このため，一度も訪れたことのないエージェント実行環境へエージェントが移動する場合であっても，他のエージェントのためにキャッシュされたプログラムコードを再利用することで，キャッシュ効率を高めることができる．

しかし，複数のエージェントが異なるバージョンのプログラムコードを利用している場合や，異なるベンダが開発したエージェントがシステム内で共存している場合に，名前が同じで動作が異なるプログラムコードを利用しているエージェントが存在する可能性がある．この場合，プログラムコードを名前で識別しようとすれば，名前が衝突してキャッシュの一貫性が失われる．そこで，プログラムコードの内容を元にプログラムコードの識別を行う．例えば，JVM の場合であれば，Java バイトコードから生成したハッシュ値をプログラムコードの識別子として利用する．このことでキャッシュの一貫性を保ちつつ他のエージェントのためにキャッシュされたプログラムコードを再利用できる．

3. エージェント移動機構

3.1 エージェントの移動セッション

本論文では，一つのエージェントの移動処理が開始してから終了するまでの一連の手続きをエージェントの移動セッションと呼ぶ．エージェントの実行時状態領域とアプリケーション領域はキャッシュの対象ではなく，移動セッションで必ず転送する必要がある．しかし，プログラムコード領域に格納されているプログラムコードはキャッシュされている可能性がある．そこで，移動セッションでは，まずエージェントの実行時状態領域，アプリケーション領域，及びプログラムコードの識別子を移動元から移動先へ転送する．移動先では，実行時状態領域とアプリケーション領域を移動対象のエージェントと関連づけて保存するとともに，プログラムコードの識別子に対応するプログラムコードがキャッシュされているかを調べる．キャッシュされていなければ，移動元へプログラムコードの転送を要求し，移動元からプログラムコードを取得する．最後に実行時状態領域，アプリケーション領域，及びプログラムコードからエージェントを復元し移動処理を終了する．以上の手続きによって，移動先にキャッシュが存在する場合はプログラムコードの転送が不要となる．

3.2 移動セッションの時間的な重なり

エージェントのプログラムコード領域は複数のプログラムコードで構成される．複数のプログラムコードを移動先にキャッシュされているか確認してから転送する場合，プログラムコードを 1 個ずつ確認して転送するよりも，複数個を一括で確認して転送する方が効果的である．また，システムの規模が大きい場合，エージェントの数も多くなり，多数のエージェントがシステム内で同時に移動するという状況が起こる．例えば，複数のエージェントが特定のエージェント実行環境に集まり，相談して意思決定するといった方法（Meeting パターン）がよく用いられる [15]．また，動的にノードが追加・削除されるような分散処理システムにおいて常に負荷が均一になるようにタスクを再分散するというモデルが提案されている [16]．このモデルでは，システムに新規ノードが追加されたときに，複数のタスクがシステム全体で均一になるよう再分散され，その際に旧ノード群から新規ノードへタスクが受け渡される．このようなモデルをモバイルエージェント技術で構築する場合，複数のエージェントが異なる実行環境から一つの実行環境へ同時に移動するときに，それぞれのエージェントの移動元へ同じプログラムコードの転送要求が重複して発行され，その結果プログラムコードが重複して転送される．このように，それぞれの移動セッションが時間的に重なることでキャッシュ効果が大きく低下する．

3.3 プログラムコード転送のキャンセル

重複したプログラムコードの転送を抑制するために，発行済みの転送要求を後からキャンセルする仕組みを導入する．キャンセルする方法では重複したプログラムコードの「転送の要求」は許容する．このため，エージェントの移動処理では，はじめに実行時状態領域，アプリケーション領域，及びプログラムコード識別子のリストを移動元から移動先へ転送する．移動先

表 1　移動セッションの手続きに用いるメッセージ
Table 1　Messages for the migration session.

メッセージ名	説明
session-open-request	移動セッションの確立を要求する．
session-open-response	移動セッションの確立を承認する．
agent	エージェントを転送する．
code-request	プログラムコードの転送を要求する．
code-response	プログラムコードを転送する．
code-cancellation	プログラムコードの転送をキャンセルする．
session-close-request	移動セッションの終了を要求する．
session-close-response	移動セッションの終了を承認する．

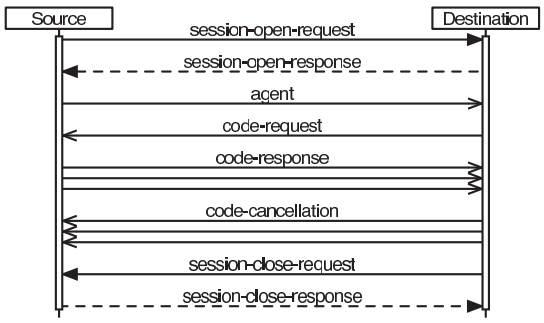


図 2　移動セッションの通信シーケンス
Fig. 2　The communication sequence diagram of an agent migration session.

では，識別子リストにあるプログラムコードについてキャッシュされているか調べ，キャッシュされていないプログラムコードの識別子リストを作成し，移動元へ転送する．識別子リストにはプログラムコードの転送を希望する順にプログラムコード識別子が並べられている．移動元は受信した識別子リストの順番に，プログラムコード送信キューへ識別子を追加する．プログラムコード送信キューは，プログラムコード識別子を要素にもつ FIFO のリストであり，移動元はプログラムコード送信キューから取り出した順番にプログラムコードを移動先へ送信する．プログラムコードを受け取った移動先は，そのプログラムコードをキャッシュし，他の移動元にそのプログラムコードを要求していないか確認する．要求していれば，それらの移動元に対してプログラムコードの転送要求をキャンセルする．キャンセルを受け取った移動元は，プログラムコード送信キューからキャンセルされたプログラムコードの識別子を削除する．これにより一つの移動先へ複数の移動元から同じプログラムコードが転送されることを抑制する．

3.4　転送プロトコルの設計

提案機構を双方向かつ非同期でメッセージを送受信可能なプロトコルとして設計する．移動セッションにおける手続きは，移動セッションの開始，エージェント及びプログラムコードの転送，移動セッションの終了に大別される．提案プロトコルでは，これらの手続きを表 1 に示す 8 種類のメッセージを用いて行う．

メッセージの通信手順を図 2 に示す．エージェントの移動セッションを確立するには，まず移動元が session-open-request メッセージを移動先に送信する．このメッセージは，複数のエージェントが同時に移動する場合を考慮して，他の移動セッションと区別するためのセッション識別子をもつ．このメッセージを受け取った移動先は，セッション識別子とともに，移動セッションの確立を承認するか否かを示すフラグを設定した session-open-response メッセージを返信する．このフラグは，エージェントの移動を承認するならば true に，しないならば false に設定される．このフラグが true の session-open-response メッセージを移動元が受信することで移動セッションが確立される．

移動セッションの確立後，エージェント及びプログラムコードの転送を行う．このために移動元は agent メッセージを移動先へ送信する．agent メッセージは，セッション識別子，エージェントの実行時状態領域のデータ，アプリケーション領域のデータ，及びプログラムコードの名前と識別子の組からなるリストで構成される．このメッセージを受信した移動先は，プログラムコードの識別子を用いて，それぞれのプログラムコードがキャッシュされているか調べる．全てのプログラムコードがキャッシュされていれば移動セッションの終了処理を行う．キャッシュされていないプログラムコードが存在すれば，キャッシュされていないプログラムコードの識別子からなるリストを生成し，code-request メッセージとして移動元へ返信する．code-request メッセージを受け取った移動元は，その中に記載されたプログラムコードの識別子のリストの先頭から順にプログラムコード送信キューに追加していく．移動元はそれと並行してプログラムコード送信キューから取り出したプログラムコードの識別子に対応するプログラムコードを code-response メッセージを用いて移動先へ返信する．code-response メッセージを受け取った移動先は，そのプログラムコードをキャッシュする．このとき，そのプログラム

コードの転送を他の移動元に対して要求中であれば，その移動元に `code-cancellation` メッセージを送信する．`code-cancellation` メッセージを受け取った移動元は，その中に記載されたプログラムコードの識別子をプログラムコード送信キューから削除する．これによりプログラムコードの送信がキャンセルされる．プログラムコードの転送が全て完了すれば，移動セッションの終了処理を行う．移動セッションを終了するためには，移動先が移動元へ `session-close-request` メッセージを送信する．このメッセージを受け取った移動元は，`session-close-response` メッセージを移動先へ返信し，移動セッションを終了する．以上の手続きにより，キャッシュによるエージェントの移動機構を実現できる．また，複数のエージェントが同時に移動する場合であっても，キャンセルを発行することで重複したプログラムコードの転送を抑制できる．

4. 実　　験

4.1 実験環境

提案手法の有効性を示すために評価実験を行う．我々が開発している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フレームワークである Maglog [17] に提案機構を実装し，DummyNet [18] による仮想的なネットワーク環境を構築して実験を行う．Maglog のエージェント実行環境は JVM 上で動作し，その上で動作するエージェントは Java インスタンスで表現されている．エージェントは，タスクの処理に必要なプログラムコードとして，Java バイトコードを保持しており，リフレクションによってそれらを動的にインスタンス化して利用する．

提案手法の実装に関して，プログラムコードの識別子には Java バイトコードから生成した SHA-1 [19] によるハッシュ値を用いる．エージェントの移動機構に用いる通信プロトコルには，HTTP に比べて低いレイテンシで双方向通信が可能な WebSocket プロトコル [20] を用いる．エージェントの移動セッションを識別するためのセッション識別子には UUID [21] を用いる．`code-request` メッセージに含めるプログラムコードの識別子のリストの順番はランダムとする．

モバイルエージェントの移動におけるプログラムコードのキャッシュの有効性を示すために，エージェントの移動機構の通信プロトコルとして以下に示す 3 種類の方式を実装する．

- キャッシュなし方式（*no-cache*）：プログラムコードをキャッシュしない方式である．プログラムコードはエージェントの移動時に必ず転送される．
- キャッシュ方式（*cache*）：プログラムコードをキャッシュし，キャンセル処理は行わない方式である．
- キャッシュ・キャンセル方式（*cache-cancel*）：プログラムコードをキャッシュし，キャンセル処理を行う方式である．

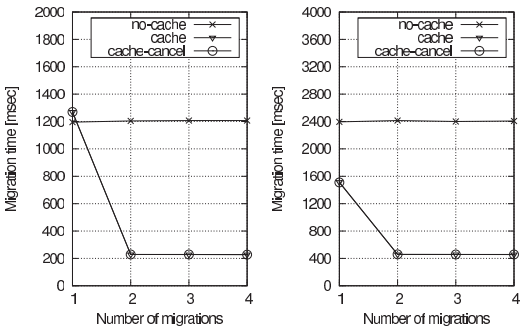
実験で用いる計算機の構成は，プロセッサが Intel®Core™i7-2600 Processor (8 MiB Cache, 3.40 GHz)，メモリが 32 GiB，HDD が 1 TiB，NIC が 1000 BASE-T，OS が Debian GNU/Linux 6.0.4 (Kernel 2.6.32-5-686-bigmem) とする．ネットワーク環境は，上記の計算機の一つの物理 NIC に IP Aliasing で複数の IP アドレスを割り当て，DummyNet [18] を用いて IP アドレス間の通信帯域を制御できるようにする．Java プラットホームには，Java™Platform, Standard Edition 6 Development Kit を用いる．

提案手法では，キャッシュしたプログラムコードを主記憶装置と補助記憶装置のどちらに保持してもよい．これらの装置の総容量は一般的にプログラムコードのデータサイズよりも十分に大きいため，キャッシュあふれ時のキャッシュ置換アルゴリズムによる影響は小さい．このため，キャッシュ方式とキャッシュ・キャンセル方式に関しては，キャッシュあふれが起こらない場合について評価する．

4.2 単体移動における評価

エージェントが単体で移動する場合におけるキャッシュの効果とキャッシュ機構のオーバヘッドを調べるために，1 個のエージェントがエージェント実行環境間を片道移動する場合と往復移動する場合における移動時間を測定した．エージェントはタスクの処理に 4096 個のプログラムコードを利用し，各プログラムコードのサイズはそれぞれ 1 KiB とした．また，エージェントの実行時状態領域は約 70 KiB，アプリケーション領域は 0 KiB とした．エージェント実行環境間は，100 BASE-T のネットワークで接続した．

測定結果を図 3 に示す．片道移動の測定結果である図 3 (a) に着目すると，キャッシュが利用できない 1 回目の移動に関してはキャッシュなし方式が最も移動時間が短い．これは，キャッシュ方式とキャッシュ・キャンセル方式では，キャッシュを利用しようとするために，プログラムコード識別子のセットを転送し，移動先でそれぞれのプログラムコードがキャッシュされているか参照するといった，キャッシュ機構のオーバヘッ



(a) 片道移動　(b) 往復移動

図 3　単体移動における移動時間
Fig. 3　Results on a single agent migration pattern.

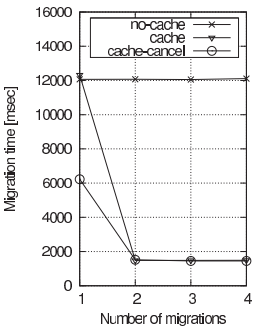


図 4　集団移動における移動時間
Fig. 4　Results on a group migration pattern.

ドが含まれているためである．プログラムコード識別子は SHA-1 によるハッシュ値であるため，データサイズは 160 bits（20 Bytes）である．本実験ではプログラムコード数が 4096 個であることから，プログラムコード識別子のセットは 80 KiB となる．このため，キャッシュなし方式では合計 4166 KiB のデータ転送が発生し，キャッシュ方式とキャッシュ・キャンセル方式では 80 KiB を加えた 4246 KiB のデータ転送が発生する．このため，キャッシュ機構がある方式では約 2%の転送量の増加となっている．また転送量の増加のほかに，プログラムコード識別子のセットの Java インスタンスを直列化・直列化復元する処理や，それに伴う Java ヒープ領域の確保といった処理もキャッシュ機構のオーバヘッドに含まれる．このように，1 回目の移動では，キャッシュ機構がある方式の方が，そのオーバヘッドにより移動時間がやや長くなってしまう．しかし，2 回目からの移動では，キャッシュが利用できるため，オーバヘッドによる移動時間の増加分よりも，キャッシュによって削減されたプログラムコード転送による移動時間の削減分の方が多いため，キャッシュなし方式よりもキャッシュ方式とキャッシュ・キャンセル方式の方が移動時間が短くなっている．一方，往復移動の測定結果である図 3 (b) に着目すると，1 回目の往復移動であってもキャッシュ機構がある方式の方が常に移動時間が短くなっている．これは，移動元にプログラムコードがキャッシュされており，復路でのプログラムコード転送が不要となるためである．なお，本実験のように単体のエージェントが移動する場合，キャンセル処理は発生しないため，キャッシュ方式とキャッシュ・キャンセル方式は，ほぼ同じ結果となっている．

4.3　集団移動における評価

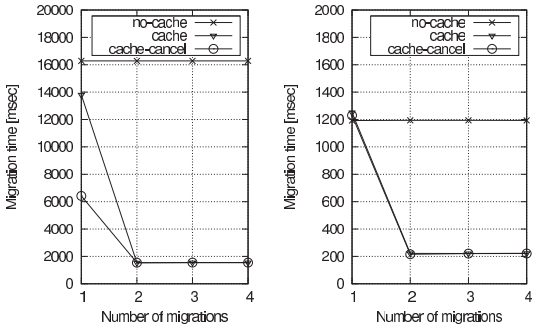
エージェントが集団で移動する場合におけるキャッシュの効果を調べるために，複数のエージェントが複数のエージェント実行環境から一つのエージェント実行環境へ同時に移動する場合における移動時間を測定した．実験では，10 個のエージェント実行環境に 1 個ずつ配置したエージェントを 1 個のエージェント実行環境へ同時に移動させた．エージェント及びエージェント実行環境は前節と同じ設定とした．

測定結果を図 4 に示す．1 回目の移動に着目すると，キャッシュ方式では同じプログラムコードが移動先のエージェント実行環境へ重複して転送されるだけでなく，キャッシュ機構のオーバーヘッドも加わるためキャッシュなし方式よりも移動時間が長くなる．しかし，キャッシュ・キャンセル方式では重複したプログラムコードの転送がキャンセルされて転送量が減少するため，移動時間が最も短くなっている．

2 回目以降の移動では，全てのプログラムコードがキャッシュされているため，キャッシュ方式とキャッシュ・キャンセル方式は同等の性能を示している．

4.4　異なる通信速度の回線が混在したネットワーク環境における評価

実際のネットワーク環境では，通信速度の異なる回線が混在している場合が多い．そこで，インターネット上に設置したエージェント実行環境に対して，3 G で接続されたエージェント実行環境と，FTTH で接続されたエージェント実行環境から，同種のエージェントが同時に移動する状況を想定した実験を行った．実験で用いるネットワークの通信帯域は，3 G 回線を 7.2 Mbit/s とし，FTTH 回線を 100 Mbit/s とした．使用するエージェント及びエージェント実行環境は **4.2** と同じものを用いた．



(a) 3 G 回線側 (7.2 Mbit/s) (b) FTTH 回線側 (100 Mbit/s)

図 5 異なる通信速度の回線が混在した環境における集団移動パターンの移動時間

Fig. 5 Results on group migration patterns on a heterogeneous network environment.

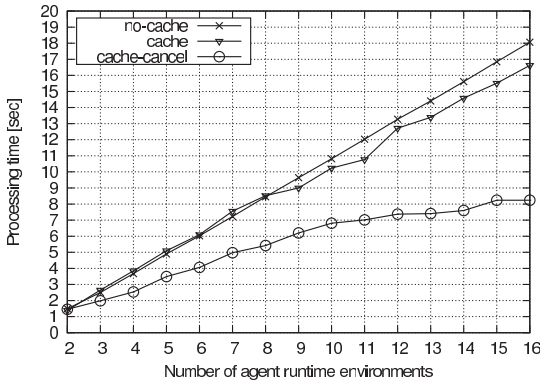


図 6 負荷分散処理に要する時間

Fig. 6 The time required for processing of a load distribution.

測定結果を図 5 に示す．3 G 回線側の 1 回目の移動では，キャッシュ・キャンセル方式は，キャッシュ方式に比べて 53.5%，キャッシュなし方式に比べて 60.6%の性能改善が見られる(注1)．キャッシュ・キャンセル方式では，プログラムコードが取得できた時点で他の全ての移動元に対してそのプログラムコードの転送をキャンセルする．このため，通信速度が 3 G 回線側に比べて高速な FTTH 回線側からプログラムコードが転送された時点で 3 G 回線側のプログラムコードの転送はキャンセルされる．その結果，3 G 回線側のエージェント移動手続きにおける多くのプログラムコード転送がキャンセルされ，移動時間を大幅に削減できた．

4.5 負荷分散システムにおける評価

実用システムにおける提案手法の有効性を評価するために，モバイルエージェントによる負荷分散システムにおける実験を行った．本実験システムでは，CPU やメモリといった計算資源の負荷を複数の計算機に分散させるために，複数のエージェントがそれぞれの計算機上のエージェント実行環境へ移動して処理を行う．それぞれのエージェントは，同じアルゴリズムを用いて問題を解くが，それぞれ異なる入力パラメータを与えられるため，実行時状態領域や，処理途中のデータであるアプリケーション領域の内容が異なる．このため，本実験システムに新しいエージェント実行環境を追加すると，既存のエージェント実行環境から新しいエージェント実行環境へそれぞれのエージェントが移動する．移動するエージェントは，移動元で一時中断していた処理を移動先で再開する．これにより，システム全体へ負荷が再分散される．

実験では，キャッシュなし方式，キャッシュ方式，及びキャッシュ・キャンセル方式の場合について，負荷分散処理が完了するまでの時間を測定した．エージェント実行環境の数は 1 台から 16 台まで順番に増加させた．ネットワークの通信帯域は 100 Mbit/s とした．

測定結果を図 6 に示す．エージェント実行環境の台数を 15 から 16 へ増設する場合において，キャッシュ方式はキャッシュなし方式に対して約 8%の性能改善が見られた．一方，キャッシュ・キャンセル方式はキャッシュなし方式に対して約 54%の性能改善が見られた．キャッシュ・キャンセル方式は，エージェント実行環境の台数が多いほどキャッシュ効率が高くなり，更にエージェント実行環境の台数を増やした場合，キャッシュなし方式とキャッシュ方式に対する性能差はより大きくなるものと考えられる．

5. 関 連 研 究

これまでに，モバイルエージェントシステムの性能を改善しようとする様々な研究が行われてきた．

文献 [6]～[8] では，ネットワークを介してリモートでのメッセージ交換を頻繁に行っているエージェント同士を同じエージェント実行環境へ移動させてローカルでのメッセージ交換にすることで，通信量を削減する手法を提案している．文献 [9] では，エージェント実行環境間で ping を送り合うことで RTT（Round-Trip

(注1)：キャッシュ方式もキャッシュ無し方式に比べてやや性能改善が見られる．これは，3 G 回線側から移動するエージェントの移動セッションのうち，(1) TCP ハンドシェイク，(2) `session-open-request` メッセージ，(3) `session-open-response` メッセージ，(4) `agent` メッセージの転送後，転送が必要なプログラムコードを決定する前に，通信速度の速い FTTH 側が既にこれらの手続きを終えて，一部のプログラムコードが転送されているからである．

Times）と平均パケットロス率を求め，エージェントがより高速な移動経路を選択しながら移動する手法が提案されている．また，移動先のエージェント実行環境で障害が発生した場合に，あらかじめ移動元に複製しておいたエージェントから経路選択をやり直すことで，エージェントの移動回数を削減し，高速な障害復帰を実現している．文献 [10] では，エージェントのサイズとメッセージのサイズを比較して移動するか，または，移動せずにリモートでのメッセージ交換だけで済ませるかを判断する手法を提案している．しかし，これらの手法 [6]〜[10] は，特定の用途に最適化したエージェントの協調アルゴリズムによる手法であるため，適用範囲が限定される．

一方，エージェントの協調アルゴリズムではなく移動機構によりモバイルエージェントシステムの性能を改善する研究はいくつか報告されている．文献 [22] では，エージェントの旅程（移動予定経路）上の実行環境に対して，エージェントが移動する前にプログラムコードを先んじて転送しておくことでエージェントの移動時間を短縮する手法が提案されている．しかし，エージェントの旅程があらかじめ分かっていることが前提であるため，移動先を動的に決定するようなエージェントには適用できない．文献 [23] では，プログラムコードを，同じセグメントに接続されている実行環境にはマルチキャストで一斉に配布し，異なるセグメントに接続されている実行環境にはセグメントを代表する実行環境へユニキャストで配布することで，通信帯域を効率良く使う手法を提案している．しかし，その適用範囲は，各セグメント内に多数の実行環境が存在し，かつネットワークが複数のセグメントから構成される場合に限定される．このように，これらの手法 [22], [23] は適用可能な範囲が特定のシステムや特定のネットワークの構成に限定される．

システムやネットワークの構成に限定される手法ではなく，より一般的に適用可能な手法として，OMG が定めた仕様である MASIF Specification [12] では，エージェントが移動するときに，プログラムコード自体ではなく，プログラムコードの名前のリストを送り，その名前を用いて移動先にプログラムコードが存在するか判定し，存在しないプログラムコードだけを一括転送する方法を挙げている．しかし，その具体的な実現方法については規定されておらず，仕様技術としては確立されていない．FIPA 仕様の実装である JADE [24] では，プログラムコードのキャッシュ機能が実装されている．しかし，異なるエージェント間でキャッシュされたプログラムコードを共有する仕組みはもたない．文献 [25] では，プログラムコードの名前の代わりに，プログラムコードの実体から生成したハッシュ値を用いることで，異なる名前空間をもった異なるエージェント間でもプログラムコードを共有可能としている．しかし，これらは単体のエージェントが単独で移動する場合しか考慮しておらず，複数のエージェントが同時に移動する場合に，同じプログラムコードを重複して転送してしまう．これに対し提案手法では，発行済みのプログラムコード転送要求をキャンセルする仕組みを導入することで，重複したプログラムコードの転送を抑制する．

6. むすび

本論文では，モバイルエージェントの移動時間を削減するためのエージェント移動機構を提案した．提案手法では，キャッシュ技術を用いることでエージェントの移動時に発生するプログラムコードの転送を削減する．また，複数のエージェントが同時に移動する場合において，重複したプログラムコードの転送をキャンセルすることでキャッシュ効率の低下を抑制する．提案手法を負荷分散システムで用いた結果，システムのエージェント実行環境の数を 1 から 16 まで拡張する処理において，最大で約 54.4%の性能向上を確認できた．

本論文では，キャッシュあふれが起こらない場合について評価したが，資源制約の強い環境下でキャッシュあふれが起こる場合についても評価することが今後の課題として挙げられる．

文　献

[1] A.R. Hurson, E. Jean, M. Ongtang, X. Gao, Y. Jiao, and T.E. Potok, “Recent advances in mobile agent-oriented applications,” Mobile Intelligence: Mobile Computing and Computational Intelligence, pp.106–139, 2010.

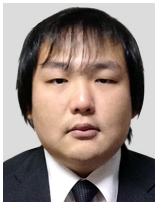
[2] A. Outtagarts, “Mobile agent-based applications : A survey,” International Journal of Computer Science and Network Security, vol.9, no.11, pp.331–339, 2009.

[3] T. Kawamura, Y. Hamada, K. Sugahara, K. Kagemoto, and S. Motomura, “Multi-agent-based approach for meeting scheduling system,” Proc. IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp.79–84, 2007.

[4] 川村尚生，菅原一孔，“モバイルエージェントに基づく p2p 型 e-learning システム，” 情処学論，vol.46, no.1, pp.222–225, 2005.

[5] 前田直人，中島　震，“分散システム開発のための移動エージェントの定量的評価手法,” 情処学論，vol.43, no.7, pp.2330–2339, 2002.
[6] 能登正人，沼澤政信，栗原正仁，“エージェントの移動性を考慮したエージェント間通信のトラフィック量に関する実験と評価,” 電学論（C），vol.124, no.3, pp.904–911, 2004.
[7] 高橋俊博，水田秀行，“大規模な分散環境での agent-based simulation フレームワーク構築における効果的なエージェント配置手法,” 情処学論「数理モデル化と応用」，vol.48, no.SIG6(TOM17), pp.120–127, 2007.
[8] 宮田直輝，石田　亨，“大規模マルチエージェントシステムにおけるエージェント配置,” 信学論（D），vol.J90-D, no.4, pp.1023–1030, April 2007.
[9] Y. Lee and K. Kim, “Optimal migration path searching using path adjustment and reassignment for mobile agent,” Proc. 4th International Conference on Networked Computing and Advanced Information Management, vol.2, pp.564–569, 2008.
[10] T.-H. Chia and S. Kannapan, “Strategically mobile agents,” Proc. 1st International Workshop on Mobile Agents, pp.149–161, 1997.
[11] A. Fuggetta, G.P. Picco, and G. Vigna, “Understanding code mobility,” IEEE Trans. Softw. Eng., vol.24, no.5, pp.342–361, 1998.
[12] Object Management Group, Inc., “Mobile agent system interoperability facilities specification,” 1997.
[13] Foundation for Intelligent Physical Agents, “Fipa agent management specification (sc00023k),” 2004.
[14] Foundation for Intelligent Physical Agents, “Fipa abstract architecture specification (sc00001l),” 2002.
[15] Y. Aridor and D.B. Lange, “Agent design patterns: Elements of agent application design,” Proc. 2nd International Conference on Autonomous Agents, pp.108–115, 1998.
[16] 須田礼仁，“Multi-master divisible load スケジューリングの最適化と漸近性能,” 情処学研報，2007-HPC-110, pp.1–6, 2007.
[17] S. Motomura, T. Kawamura, and K. Sugahara, “Logic-based mobile agent framework with a concept of “field”,” IPSJ J., vol.47, no.4, pp.1230–1238, 2006.
[18] L. Rizzo, “Dummynet: A simple approach to the evaluation of network protocols,” ACM SIGCOMM Computer Communication Review, vol.27, pp.31–41, 1997.
[19] D. Eastlake 3rd and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” Request for Comments 3174, 2001.
[20] I. Fette and A. Melnikov, “The WebSocket Protocol,” Request for Comments 6455, 2011.
[21] P. Leach, M. Mealling, and R. Salz, “A Universally Unique IDentifier (UUID) URN Namespace,” Request for Comments 4122, 2005.
[22] G. Soares and L.M. Silva, “Optimizing the migration of mobile agents,” Proc. 1st International Workshop on Mobile Agents for Telecommunication Applications, pp.161–178, 1999.
[23] D. Gavalas, “An experimental approach for optimising mobile agent migrations,” Mediterranean Journal of Computers and Networks, vol.1, no.1, pp.47–56, 2005.
[24] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa, “Jade programmer’s guide,” 2010.
[25] P. Braun and W. Rossak, Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit, Morgan Kaufmann Publishers 2005.

（平成 24 年 11 月 7 日受付，25 年 2 月 16 日再受付）



東野　正幸

平 22 鳥取大学大学院工学研究科情報エレクトロニクス専攻博士前期課程了．現在，同専攻博士後期課程在学中，リサーチアシスタントとして従事．エージェントシステム，情報システムに興味をもつ．情報処理学会，ACM，IEEE 各学生会員．

高橋　健一　（正員）

平 16 九州大学大学院システム情報科学府博士課程了．博士（工学）．同年，財団法人九州システム情報技術研究所（現，九州先端科学技術研究所）入所．平 23 より，鳥取大学大学院工学研究科情報エレクトロニクス専攻准教授．情報セキュリティ，エージェントシステム，ユビキタス技術等の研究に従事．情報処理学会，電気学会各会員．

川村　尚生　（正員）

平 6 神戸大学大学院自然科学研究科博士課程単位取得退学．同年鳥取大学工学部知能情報工学科助手，現在，同大学大学院工学研究科情報エレクトロニクス専攻教授．エージェントシステム，社会情報システムに関する研究に従事．博士（工学）．情報処理学会会員．

菅原　一孔　（正員）

昭 56 東京工業大学大学院理工学研究科電子物理工学専攻修士課程了．同年神戸市立工業高等専門学校電気工学科講師．同校助教授を経て平 6 鳥取大学工学部電気電子工学科助教授，現在，同大学大学院工学研究科情報エレクトロニクス専攻教授．計算機工学に関する研究に従事．工学博士．IEEE，情報処理学会各会員．