GAPを用いたキャッシュによるエージェントの同時集中移動時における通信量の削減

東野 正幸　高橋 健一　川村 尚生　菅原 一孔

モバイルエージェントとはノード間を移動できる自律的なソフトウェアである．モバイルエージェントは移動先にプログラムコードを持ち運ぶことで移動元で行っていた処理を移動先でも継続できる．モバイルエージェントシステムでは，複製などの処理によって異なるモバイルエージェントが同じプログラムコードを持っている場合があり，そのようなモバイルエージェントが複数同時に移動しようとすると，キャッシュミスが同時に発生することで同じプログラムコードが重複して転送される問題がある．本論文では，このような問題を一般化割当問題 (generalized assignment problem: GAP) として定式化し，重複したプログラムコードの転送を抑制するモバイルエージェントの移動方式を提案する．

A mobile agent is autonomous software which can migrate among different nodes. A mobile agent can continue with the execution of their task before and after migration among different nodes with transferring of program codes. In a mobile agent system, cloned mobile agents have same program codes. When such mobile agents migrate from different nodes to same one node, a duplicate transfer of same program codes occur, and the data traffic increases. This paper formulates this problem as the generalized assignment problem (GAP) and proposes a mechanism of agent migration to prevent duplicate transfer of same program codes.

1　はじめに

モバイルエージェントとはノード間を移動可能な自律的なソフトウェアである．モバイルエージェント技術は，分散アプリケーションの設計パラダイムとして注目され，通信の効率化や複雑な設計の簡素化など幅広い分野で利用されている [6] [9]．モバイルエージェントはノード間を移動する際に処理対象のデータだけでなく処理に利用するプログラムコードやプログラムの実行時状態の転送を必要とするためモバイルエージェントの移動は通信量を増加させる．このため，モバイルエージェントの移動時にプログラムコードをノードにキャッシュすることで通信量や通信時間を削減する移動方式が提案されている [8] [4] [12]．

モバイルエージェントシステムでは，複数のエージェントが特定のノードに集まり，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 [2]．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モデルが提案されている [13]．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，旧ノード群から新規ノードへ複数のエージェントが同時に移動し，その同時移動数はノード数に比例して増大する．

一般にソフトウェア開発ではプログラムコードの再利用が行われる．モバイルエージェント技術で構築

Reduction of Traffic for Mobile Agent Migration using GAP.

Masayuki Higashino, 鳥取大学産学・地域連携推進機構, Organization for Regional Industrial Academic Cooperation, Tottori University.

Kenichi Takahashi, Takao Kawamura, Kazunori Sugahara, 鳥取大学大学院工学研究科情報エレクトロニクス専攻, Graduate School of Engineering, Tottori University.

コンピュータソフトウェア, Vol.31, No.3 (2014), pp.168–177.

[研究論文] 2013 年 7 月 24 日受付.

されたソフトウェアにおいてもプログラムコードの再利用により異なるエージェントが同じプログラムコードを持つ．例えば，エージェントが複製された場合や，異なるエージェントが同じライブラリを持っている場合である．しかし既存の移動方式では，複数のエージェントが異なるノードから 1 つのノードへ短時間に集中して移動 (以降，同時集中移動と呼ぶ.) する場合にキャッシュミス (エージェントが持つ，あるプログラムコードが移動先ノードにキャッシュされていない状態) が同時に発生し，同じプログラムコードが重複して転送される問題や，プログラムコード数が多い場合にプログラムコードの転送制御用メッセージの通信量が増加するといった問題がある．

そこで本論文では，同時集中移動しようとする各々のエージェントが同じプログラムコードを持っている場合に，組み合わせ最適化問題の一種である一般化割当問題 (generalized assignment problem: GAP) として各々のプログラムコードの転送元ノードを決定する方式を提案する．本方式により，各々のプログラムコードの転送元ノードをいずれか 1 つに決定して転送することで，同じプログラムコードの重複した転送を抑制できる．

以降の論文構成を述べる．2 節では関連研究を示すと共に既存方式の問題点を示す．3 節では，提案手法が対象とするモバイルエージェントのモデルを説明する．4 節では提案手法について述べる．5 節では提案手法の評価を行う．6 節で本論文をまとめる．

2 関連研究

Object Management Group (OMG) が定めた仕様である MASIF Specification [8] では，エージェントが移動する際に，まず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を一括で転送する手法を挙げている．しかし，その具体的な実現方法は規定されていない．

Aglets [1]，Mobile Code Toolkit [7]，及び Foundation for Intelligent Physical Agents (FIPA) の仕様の実装である JADE [3] 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，キャッシュされたプログラムコードを異なるエージェント間で共有する仕組みを持たないため，別のエージェントの移動時にキャッシュされたプログラムコードを，他のエージェントの移動時にキャッシュとして利用する事はできない．

Kalong [4] では，プログラムコードの名前の代わりに，プログラムコードから生成したハッシュ値を用いてプログラムコードを識別することで，異なるエージェント間でもプログラムコードのキャッシュを共有可能にしている．エージェントが移動する際は，まずハッシュ値のリストを送り，そのハッシュ値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で転送する．しかし，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，キャッシュミスが同時に発生することで，同じプログラムコードが重複して転送されてしまう．

文献 [12] では，同時集中移動時のキャッシュミスの同時発生によって移動先ノードから各々の移動元ノードへ同じプログラムコードの転送要求が行われた場合に，各々の移動元ノードから行われつつある同じプログラムコードの転送について，何れか 1 つの移動元ノードからの転送が完了した時点で，移動先ノードが他の移動元ノードにキャンセル要求を行うことで，重複したプログラムコードの転送を抑制する手法が提案されている．しかし，プログラムコード数が増加するとキャンセル要求による通信量も増加してしまう．

そこで本論文では，キャッシュされたプログラムコードを異なるエージェント間で共有可能とし，さらに同時集中移動時に各々のプログラムコードの転送元ノードをいずれか 1 つに決定して転送することで，同じプログラムコードの重複した転送を抑制する方式を提案する．

3 モバイルエージェントのモデル

3.1 エージェントの構造

エージェントの移動を実現するためにはエージェントを構成するデータを移動元ノードから移動先ノードへ転送する必要がある．そこでまずエージェントの構造を定義する．モバイルエージェントの構造を図

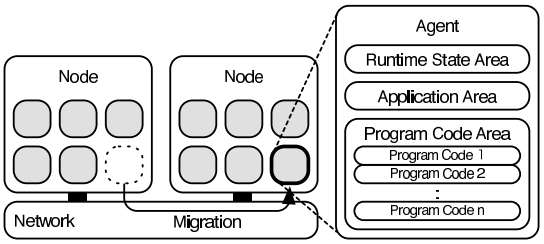

図 1　エージェントの構造

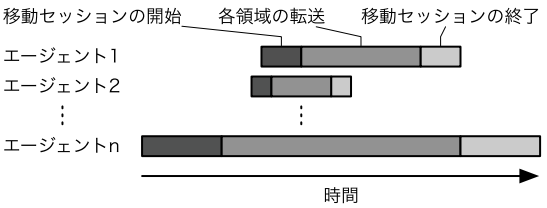

図 2　移動セッションの時間的な重なり

1 に示す．ノードでは複数のエージェントが並列に動作し，これらのエージェントはネットワークを介してノード間を移動できる．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から構成される [5]．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，動作中の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情報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複数のプログラムコードを保持する．

3.2　エージェントの移動

本論文では，1 つのエージェントの移動処理を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．移動セッションは，移動セッションの開始，エージェントを構成する各領域の転送，移動セッションの終了，に大別される．移動セッションの開始では，移動元ノードが移動先ノードへエージェントの移動の許可を申請し，許可されればエージェントを構成する各領域の転送準備を行う．エージェントを構成する各領域の転送では，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を転送する．移動セッションの終了では，移動先ノードが移動元ノードへエージェントを構成する各領域が正常に転送完了したか否かを通知した後にエージェントの移動を完了する．

3.3　エージェントの同時集中移動

複数のエージェントが異なるノードから 1 つのノードへ短時間に集中して移動することで複数の移動セッションが時間的に重なる事をエージェントの同時集中移動と呼ぶ．同時集中移動で複数の移動セッションが時間的に重なる例を図 2 に示す．この例では，同じプログラムコードのセットを持つ異なるエージェント $\{a_1, a_2, ..., a_n\}$ が，異なるノードから 1 つのノードへ同時集中移動する場合を示している．ただし，それぞれの移動元ノードと移動先ノードとの通信速度は $\{v_1, v_2, ..., v_n : v_2 > v_1 > v_n\}$ とする．通信速度が速いほど移動セッションの開始，各領域の転送，移動セッションの終了の処理に要する時間が短くなるため，移動セッションの開始から終了までの時間は $a_n > a_1 > a_2$ となる．また，各エージェントの移動セッションの開始時刻には差が有り，各々のエージェントは互いの移動開始時刻を知らない．

分散アプリケーションにおいてこのようなエージェントの同時集中移動は様々な状況で発生する．例えば，モバイルエージェントのためのデザインパターンの 1 つである Meeting パターン [2] では，複数のエージェントの意思を合成するために特定のノードにエージェントが集合する．また，エージェントの協調に使われる黒板モデルでは，黒板が配置されているノードにエージェントが集中する場合がある．また，マルチエージェントシステムにおいてはエージェントの集団的な振舞いの変化が相転移現象を引き起こしシステム性能を突如不連続的に変化させる可能性があることが示されており [11] エージェントが移動性を持つ場合では短時間で複数のエージェントが集団的に移動することが考えられる．

このような場合，Kalong [4] の移動方式 (以降，キャッシュ方式と呼ぶ.) では，プログラムコードの転送の直前に移動先ノードにキャッシュが存在するか

まとめて確認するが，確認する時点で他の移動セッションで転送されていないプログラムコードについてはキャッシュミスとなり重複して転送されてしまう．文献[12] の移動方式 (以降，キャンセル方式と呼ぶ.) では，キャッシュミスの同時発生により各々の移動元ノードに同じプログラムコードの転送要求が発行されることで同じプログラムコードが重複して転送されつつある場合に，ある移動元ノードから移動先ノードへプログラムコードが到達した時点で他の移動元ノードに転送のキャンセル要求を発行することで，プログラムコードの重複した転送を抑制するが，プログラムコード数が増加するとキャンセル要求による通信量も増加してしまう．

そこで本論文では，どのプログラムコードをどの移動元ノードに転送要求したかを移動先ノードに記憶し，各々の移動セッションでプログラムコードを転送する前に，時間的に重なっている移動セッション全体を考慮してプログラムコードの転送元ノードをいずれか 1 つに決定する方式 (以降，本方式を要求割当方式と呼ぶ.) を提案する．これにより，キャンセル要求が不要となり，エージェントが持つプログラムコード数が増加してもキャンセル要求による通信量は増加しない．

4 提案方式

4.1 プログラムコードの転送可否

エージェントが自身の動作を変更する場合に自身が保持しているプログラムコードを追加・削除・変更する事が考えられる．このため，同時集中移動する各々のエージェントが持っているプログラムコードの集合は，完全に一致しているとは限らず，部分的に一致している場合も考慮する必要がある．したがって，同時集中移動時にプログラムコードの転送元ノードを決定する場合，プログラムコードを取得可能な移動セッションと取得不可能な移動セッションを識別し，プログラムコードを取得可能な移動セッションから転送元ノードを 1 つ選択する必要がある．このような条件は次式で表現できる．

$$\sum_{i=1}^{m} x_{ij} = 1,\ \forall i \in I,\ \forall j \in J,\ x_{ij} \in \{0,1\} \quad (1)$$

ここで，$I = \{1, ..., m\}$ は時間的に重なった移動セッションの移動元ノード集合，$J = \{1, ..., n\}$ は時間的に重なった移動セッション全体で転送する必要がある n 個の重複なしプログラムコード集合，x_{ij} は移動元 i からプログラムコード j を転送する場合に 1，そうでない場合に 0 の値をとる 0 – 1 変数である．

4.2 通信速度の差異

一般的にコンピュータネットワークではノードの接続形態の違いやノード間の経路の違いによってノード間の通信速度に差が存在する．通信速度の違いを考慮せずにプログラムコードの転送元ノードを選択した場合，通信速度の遅い移動セッションでのプログラムコード転送がボトルネックになる．このため，各移動セッションでの通信速度と転送するプログラムコードの総データサイズが比例するように，プログラムコードの転送元ノードを選択する必要がある．このような条件は次式で表現できる．

$$s_i = \frac{\sum_{j=1}^{n} a_j}{\sum_{i=1}^{m} b_i} b_i \quad (2)$$

ここで，a_j はプログラムコード j のデータサイズ，b_i は移動元ノード i と移動先ノードとの通信速度，s_i は移動元ノード i から移動先ノードへのプログラムコードの転送に利用して良い通信量である．

また，一般的にプログラムコードのデータサイズはそれぞれ異なるため，多くの場合で，移動元ノード i から移動先ノードへのプログラムコードの転送に利用して良い通信量 s_i と，その移動元ノード i から転送するプログラムコードのデータサイズの和を完全に等しくすることはできない．そこで以下の条件により，移動元ノード i から転送するプログラムコードのデータサイズの和がその移動元ノード i からプログラムコードの転送に利用してよい通信量 s_i を超えないようにする．このような条件は次式で表現できる．

$$s_i \geq \sum_{j=1}^{n} a_j x_{ij} \quad (3)$$

4.3 一般化割当問題に対する提案方式の適用

式 1，式 2，及び式 3 の条件の下でプログラムコードの転送元ノードを決定する．ここで，プログラムコード j を移動元ノード i から転送できる場合に 0，そうでない場合に ∞ のコストを要すると考えれば，総コストが 0 となるように目的関数を設定すれば良い．これは次式の最適化問題として表現できる．

$$\begin{aligned} \text{最小化} \quad & cost = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{制約条件} \quad & s_i \geq \sum_{j=1}^{n} a_j x_{ij}, \qquad (4) \\ & \sum_{i=1}^{m} x_{ij} = 1, \\ & c_{ij} \in \{0, \infty\}, \\ & x_{ij} \in \{0, 1\}, \\ & \forall i \in I,\ \forall j \in J \end{aligned}$$

ここで，$cost$ は目的関数，c_{ij} は移動元ノード i からプログラムコード j を転送できる場合に 0，そうでない場合に ∞ の値をとる変数である．この式において $cost$ が最小となる x_{ij} 変数を決定することを目的とする．

このような問題は一般化割当問題 (generalized assignment problem: GAP) として解く事ができる．GAP とは，n 個の仕事を m 個の機械のどれかに全て割当てる問題で，機械が仕事を行った時に負荷がかかるとし，各機械に割当てられた仕事の負荷の総和が，ある機械の能力の制限を越えないように割当てる制約がある問題である．

4.4 制約条件の緩和

一般的にプログラムコードのデータサイズはそれぞれ異なる．例えば，Java Runtime Environment (JRE) のシステムライブラリ中のバイトコードでは，最小で 102 bytes，最大で 131311 bytes となっており，その差は約 1287 倍にも及ぶ†1．このため，データサイズの大きなプログラムコードが低速な通信速度で接続されている移動元ノードからしか転送できない場合，移動元ノード i からプログラムコードの転送に利用しても良い通信量 s_i を超過せざるを得ず，制約条件の式 4 を満たせない場合が発生する可能性がある．

そこで提案方式では制約条件の緩和を検討する．式 4 を制約条件から除き，代わりに移動元ノード i から移動先ノードにおいてプログラムコードの転送に利用しても良い通信量 s_i と，移動元ノード i から移動先ノードへ実際に転送するプログラムコードのデータサイズの和との差が 0 に近づく様にする．制約条件を緩和した式を以下に示す．

$$\begin{aligned} \text{最小化} \quad & cost = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ & \quad + \sum_{i=1}^{m}\left|\sum_{j=1}^{n} a_j x_{ij} - s_i\right| \qquad (5) \\ \text{制約条件} \quad & \sum_{i=1}^{m} x_{ij} = 1, \\ & c_{ij} \in \{0, \infty\}, \\ & x_{ij} \in \{0, 1\}, \\ & \forall i \in I,\ \forall j \in J \end{aligned}$$

4.5 要求割当に用いるアルゴリズム

GAP は NP 困難である事が知られている [10]．エージェントの同時集中移動は，複数のエージェントが異なるノードから 1 つのノードへ短時間に集中して移動することから，要求割当は少ない計算量で行われる事が望ましい．そこで，少ない計算量で GAP の近似解を得る事ができる解法として，欲張り法 (greedy algorithm) を利用する．エージェントの同時集中移動数を m，重複無しのプログラムコード数を n とすると，最適解を得られる列挙法では $O(m^n)$ の計算量を要するのに対して，欲張り法は $O(mn)$ で近似解を得る事ができる．

提案方式では以下の手続きで各プログラムコードの転送元を決定する．

Step 1 式 2 を用いて全ての s_i を計算する．

Step 2 式 5 を用いて全ての i と j の組合せについてコストを計算し，これらの集合を候補リストと

†1 Mac OS X, Java™SE Runtime Environment (build 1.6.0_45-b06-451-11M4406), file:///System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/classes.jar.

する．

Step 3 コストの最も低い i と j の組を採用する．

Step 4 採用した j と同じ j を用いた他の i と j の組を候補リストから削除する．

Step 5 採用した i と j について，s_i から a_j を減じる．

Step 6 候補リストが空であれば終了する．そうでなければ Step 2 を実行する．

4.6 要求割当のタイミング

エージェントが同時集中移動する時，各々の移動セッションの開始時刻は少しずつ異なる場合がほとんどである．したがって，例えば通信速度の遅い移動セッションが開始された後で，通信速度の速い移動セッションが開始された場合，後から開始された移動セッションでより多くのプログラムコードを転送した方が転送時間を削減できる．また，移動セッションが途中で切断される可能性もある．この場合，切断された移動セッションで転送を予定していたプログラムコードを別の移動セッションから転送するようにしなければならない．そこで提案方式では，同時集中移動時において移動セッションの重なり数が増減する毎に要求割当を行う．

5 評価

同時集中移動時における提案方式の有効性を評価するために既存方式と提案方式の比較実験を行う．比較する各方式の特徴を以下に示す．

キャッシュ無し方式 プログラムコードをキャッシュしない方式である．

キャッシュ方式 プログラムコードをキャッシュする方式である．また，異なるエージェント間でキャッシュを共有できる．ただし，同時集中移動時に同じプログラムコードが重複して転送されてしまう．この方式は文献[4] の方式に該当する．

キャンセル方式 キャッシュ方式にプログラムコード転送のキャンセル機能を加えた方式である．キャンセル機能により同時集中移動時の重複したプログラムコードの転送を抑制できる．ただし，プログラムコードの数とプログラムコードの共有率が増加すると，同じプログラムコードの重複した転送をキャンセルするためのメッセージの通信量が増加してしまう．この方式は文献[12] の方式に該当する．

要求割当方式 本論文の提案方式である．キャッシュ方式にプログラムコードの要求割当機能を加えた方式である．プログラムコードの転送元ノードを予め決定してから転送するため，プログラムコードの数とプログラムコードの共有率が増加しても，キャンセルするためのメッセージの通信量は増加しない．

実験では，数値解析によりこれらの方式を比較評価した．以降の評価に用いるパラメータを表 1 に示す．また，キャッシュ方式，キャンセル方式，及び要求割当方式では，各プログラムコードが移動先ノードにキャッシュされているかを確認するために，プログラムコードのデータに対する SHA-1 によるハッシュ値を用いて移動先ノードに問い合わせることとした．また本実験では，各移動方式および変化させるパラメータの組み合せに対して 10 回の測定を行ない，その転送量の平均値を評価した．

5.1 同時移動数に対する転送量の評価

同時移動数が多い場合における提案方式の有効性を示す為に，同時移動数に対する転送量の変化を比較した．

実験結果を図 3 に示す．キャッシュ無し方式では，プログラムコードをキャッシュしないため，エージェントの同時移動数に比例してプログラムコードの転送量が増加している．キャッシュ方式では，同時集中移動時に各移動セッションでキャッシュミスが同時に発生することで，プログラムコードが重複して転送されている．なお，同時移動数が増えるにつれてキャッシュ無し方式よりも転送量が減少しているのは，各移動セッションの開始時刻が 70 ミリ秒ずつずれているためである．同時移動数が多い場合，移動セッションの開始時刻が後の移動セッションが開始された時には，既にいくつかのプログラムコードが転送されているため，後半の移動セッションほどキャッシュミス率は低くなっていく．しかし，各移動セッションの開

表 1　評価に用いるパラメータ

	5.1 節	5.2 節	5.3 節	5.4 節	5.5 節
同時移動数 [個]	[1, 20]	10	10	10	10
通信速度 [MBit/s]	100	{7.2, 8.0, 12.0, 100.0}	100	100	100
プログラムコード数 [個]	4096	4096	[500, 5000]	4096	4096
プログラムコードサイズ [KB]	1	1	1	[1, 16]	1
プログラムコード共有率 [%]	100	100	100	100	[0, 100]
開始時間のずれ [ms]	70	70	70	70	70
アプリケーション領域のデータサイズ [KB]	0	0	0	0	0
実行時状態領域のデータサイズ [KB]	72	72	72	72	72

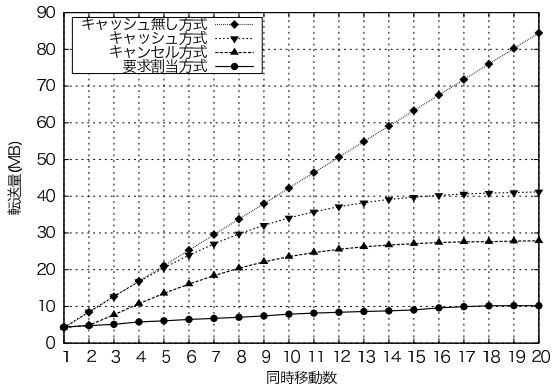

図 3　同時移動数に対する転送量の変化

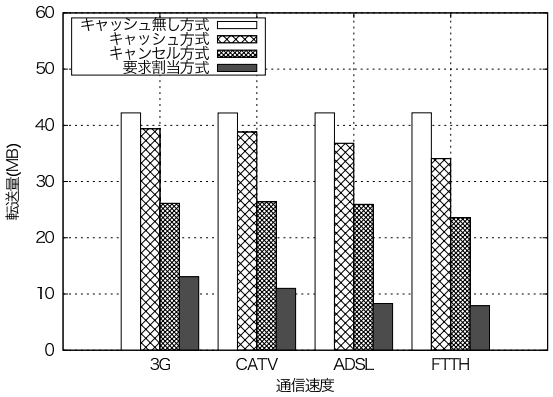

図 4　通信速度に対する転送量の変化

始時刻が更に集中すればキャッシュミス率はより高くなると考えられる．キャンセル方式では，キャッシュミスにより生じたプログラムコードの転送要求を後からキャンセルすることで転送量を抑制している．しかし，同時移動数が 20 の場合にキャッシュ方式に対して 33%程度しか通信量を削減できていない．一方，要求割当方式では 75%削減できている．なお，要求割当方式においても，同時移動数に比例して転送量が増加しているのは，各エージェントが持つアプリケーション領域，実行時状態領域，及び各プログラムコードのハッシュ値の集合の転送によるものである．これらはキャッシュされないためエージェントの移動時に必ず転送される．

5.2　通信速度に対する転送量の評価

通信速度が異なる環境での提案方式の有効性を示すために，通信速度に対する転送量の変化を比較した．通信速度の設定には，実際に存在する通信環境を想定して，3G (7.2 Mbit/s), CATV (8 Mbit/s), ADSL (12 Mbit/s), FTTH (100 Mbit/s) を用いた．

実験結果を図 4 に示す．実験では，どの通信速度においても要求割当方式が転送量を最も抑制できている．キャッシュ無し方式では，プログラムコードをキャッシュしないため，どの通信速度においても転送されるプログラムコードの量は同じである．一方，キャッシュ方式，キャンセル方式，及び要求割当方式のようにキャッシュ機能をもつ方式では，通信速度が速くなるほど転送量が少なくなっている．これは，通信速度が速いほど移動開始時刻のずれの時間中に転送できるプログラムコードの量が多くなるため，同時集中移動数がより少ないうちにより多くのプログラムコードの転送が完了していることを表している．つまり，通信速度が速いほど同時集中移動数が少ないうちにより多くのプログラムコードの転送が完了す

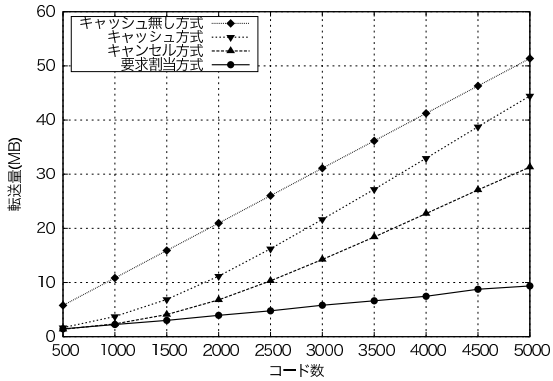

図 5 プログラムコードの数に対する転送量の変化

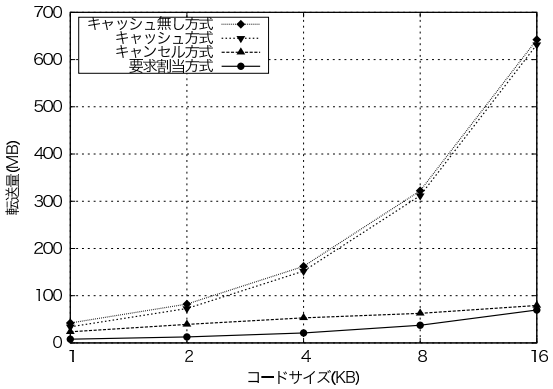

図 6 プログラムコードのデータサイズに対する転送量の変化

るため，同時集中移動の終盤におけるキャンセル処理や再割当処理によって途中で転送が中断された分のプログラムコードの転送量が減少している．

5.3 プログラムコードの数に対する転送量の評価

プログラムコードの数に対する転送量の変化を比較した．実験では，プログラムコードの数を 500 から 5000 まで 500 ずつ増加させた．

実験結果を図 5 に示す．キャッシュ無し方式では，同時移動数が 10 であることから，プログラムコードの数が 500 増える毎に，転送量が約 5 MB ずつ増加している．キャッシュ方式は，同時集中移動を考慮していないため，転送量をほとんど削減できていない．キャンセル方式は，キャッシュ方式よりも転送量を抑制できているものの，プログラムコード数の増加と共にキャンセル要求によるメッセージの転送量が増加している．要求割当方式では，プログラムコード数が増加しても割当制御に要するメッセージの数は増加せず，転送量を他の方式よりも抑えられている．

5.4 プログラムコードのデータサイズに対する転送量の評価

プログラムコードのデータサイズに対する転送量を比較した．実験では，各プログラムコードのデータサイズを 1 KB から 16 KB まで増加させた．

実験結果を図 6 に示す．実験では要求割当方式が最も転送量が抑えられている．キャッシュ無し方式とキャッシュ方式とでは，データサイズが大きくなっても転送量にほぼ差がない．これは，データサイズの増加によって，プログラムコードの転送時間が増加し，プログラムコードの要求時にキャッシュされているプログラムコードの数が減っているためである．また，コードサイズが大きくなるほど，キャンセル方式の転送量が要求割当方式に近づいている．これは，プログラムコードのデータサイズが大きくなるほど，プログラムコードの転送に時間を多く要するため，キャンセル方式におけるキャンセル要求が移動元ノードに素早く到達してキャンセル処理が間に合う可能性が高くなり，重複した転送が起こりにくくなるためである．

5.5 プログラムコードの共有率に対する転送量の評価

キャッシュ機能を持つ移動方式の場合，同時集中移動する各エージェントが同じプログラムコードを保持しているほど転送量をより多く削減できることが望まれる．そこで，プログラムコードの共有率に対する転送量を比較した．同時集中移動する各々のエージェントが持っているプログラムコードの集合のうち，共通部分が無い場合は共有率が 0%となり，全て等しい場合は共有率が 100%となる．

実験結果を図 7 に示す．キャッシュ無し方式ではキャッシュ機能が無いため転送量は一定で最も多くなっている．一方，キャッシュ機能を持つその他の移動方式ではプログラムコードの共有率に比例して転送量が減少している．キャッシュ方式では，各移動セッションが開始された時点で，前の移動セッションによ

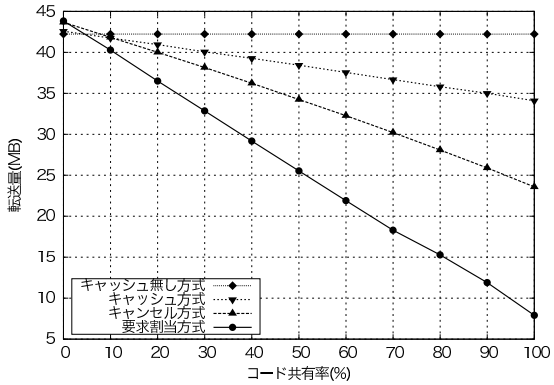

図 7　プログラムコードの共有率に対する転送量の変化

るキャッシュ済みのプログラムコードの量が共有率に比例して増加するため，プログラムコードの共有率に比例して転送量が減少している．キャンセル方式では，プログラムコードの共有率に比例して転送がキャンセルされるプログラムコードの数が比例するため，キャッシュ方式よりも転送量の削減比率が高くなっている．要求割当方式では，移動セッションの重なりが増える毎にプログラムコードの転送元ノードを決定しているため，他の移動方式に比べてプログラムコードや制御用メッセージの無駄な転送が抑制され，転送量の削減比率が最も高くなっている．

6 まとめ

本論文では，複数のモバイルエージェントが異なるエージェント実行環境から 1 つのエージェント実行環境へ集中して同時に移動する場合における通信量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，同時に移動しようとする異なるエージェントが同じプログラムコードを持っている場合に，各々のプログラムコードの転送元ノードをいずれか 1 つに決定して転送することで，同じプログラムコードの重複した転送を抑制する．従来手法と提案手法に対して様々なパラメータを用いて計算実験を行ったところ，エージェントの同時移動数が多い場合，プログラムコードの数が多い場合，プログラムコードのデータサイズが大きい場合，エージェント間でのプログラムコードの共有率が高い場合について，従来手法よりも良い結果を得る事ができた．今後の課題として通信時間による評価や多種多様なエージェント及び通信環境が混在した環境での評価が挙げられる．

謝辞 本研究において議論や実験に協力いただいた新元悠起氏に感謝します．

参 考 文 献

[1] Aglets, http://aglets.sourceforge.net/, 2013.
[2] Aridor, Y. and Lange, D. B.: Agent Design Patterns: Elements of Agent Application Design, in *Proceedings of the 2nd International Conference on Autonomous Agents*, 1998, pp. 108–115.
[3] Bellifemine, F., Caire, G., Trucco, T. and Rimassa, G.: *Jade Programmer's Guide*, 2010.
[4] Braun, P. and Rossak, W.: *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*, Morgan Kaufmann Publishers Inc., 2005.
[5] Fuggetta, A., Picco, G. P. and Vigna, G.: Understanding Code Mobility, *IEEE Trans. on Softw. Eng.*, Vol. 24, No. 5(1998), pp. 342–361.
[6] Hurson, A. R., Jean, E., Ongtang, M., Gao, X., Jiao, Y. and Potok, T. E.: *Recent advances in mobile agent-oriented applications*, John Wiley & Sons, Inc., 2010, pp. 106–139.
[7] Mobile Code Toolkit, http://www.sce.carleton.ca/netmanage/mctoolkit/, 2013.
[8] Object Management Group, Inc.: *Mobile Agent System Interoperability Facilities Specification*, 1997.
[9] Outtagarts, A.: Mobile Agent-based Applications : a Survey, *International Journal of Computer Science and Network Security*, Vol. 9, No. 11(2009), pp. 331–339.
[10] Sahni, S. and Gonzalez, T.: P-Complete Approximation Problems, *Journal of the ACM*, Vol. 23, No. 3(1976), pp. 555–565.
[11] 原田耕治, 木下哲男, 白鳥則郎: マルチエージェントパフォーマンス制御に於ける相転移の役割, 電子情報通信学会技術研究報告, Vol. 103, No. 244(2003), pp. 19–23.
[12] 東野正幸, 高橋健一, 川村尚生, 菅原一孔: キャッシュによるエージェントの移動効率化, 電子情報通信学会論文誌, Vol. J96-D, No. 7(2013), pp. 1576–1584.
[13] 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, 情報処理学会研究報告, Vol. 2007, No. 59(2007), pp. 1–6.

東 野 正 幸

1983 年生．2014 年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後期課程修了．博士 (工学)．同年鳥取大学産学・地域連携推進機構

プロジェクト研究員．エージェントシステム，情報システムに興味を持つ．情報処理学会，ACM，IEEE 各会員．

高橋健一

1975 年生．2004 年九州大学大学院システム情報科学府博士課程修了．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．2011 年より，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．電子情報通信学会，情報処理学会，電気学会，IEEE 各会員．

川村尚生

1965 年生．1994 年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)．電子情報通信学会，情報処理学会各会員．

菅原一孔

1956 年生．1981 年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修了．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 1994 年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．電子情報通信学会，情報処理学会各会員．