A-008

Erlangを用いたマルチエージェントシミュレーションのための基礎的研究

川上大輔† 松井俊浩† 松尾啓志†

近年、通信網を介した分散システムの大規模化、高度化を背景に、応用的なマルチエージェントシステムが提案されている。しかし、従来のプログラミング言語では、非同期的なメッセージ通信型プログラミングや協調プロトコルの記述などが煩雑になるなど、分散・協調処理の実装におけるプログラムの記述性に問題がある。また、多数のエージェントを同時に動作させることが困難な場合があるなど、スケーラビリティにも問題がある。そこで、本研究では関数型プログラミング言語の一つである、Erlangに着目した。マルチエージェントシステムの実装におけるErlangの有効性を評価・検討するため、ErlangとJavaをそれぞれ用いたマルチエージェントシミュレーションプログラムを実装し、比較実験を行った。その結果、性能、及び記述性の面でErlangに有効性が見られることが確認できた。また、Erlangを用いたマルチエージェントシミュレーションシステムを実装し、Erlangによるマルチエージェントシステムの実装における問題点を解決することを検討した。

A Study of Multi Agent Simulation using Erlang

Daisuke Kawakami,† Toshihiro Matsui† and Hiroshi Matsuo†

1. はじめに

近年、通信網を介した分散システムの大規模化、高度化を背景に、応用的なマルチエージェントシステムが提案されている。しかし、一般的に用いられるCやJavaなどのプログラミング言語では、非同期的なメッセージ通信型プログラミングや協調プロトコルの記述などが複雑になるなど、記述性に問題がある。また、多数のエージェントを同時に動作させることが困難な場合があるなど、スケーラビリティにも問題がある。以上のような理由から、並列分散システムのプログラミングに有効な言語が求められている。一方で、分散処理のプログラミングに適した言語の一つとして、Erlang[1)]が注目されている。Erlangは、関数型プログラミング言語の一つで、ステートレス、プロセスが非常に軽量、分散並列処理プログラムが比較的容易に記述できるといった特徴がある。このような特徴はマルチエージェントシステムの実装に適していると考えられる。そこで、本研究ではErlangを用いたマルチエージェントシステムを実装することで、その有効性を評価・検討した。

2. マルチエージェントシステム

本研究の対象とするマルチエージェントシステムの説明を示す。

† 名古屋工業大学
Nagoya Institute of Technology

2.1 マルチエージェントシステムの概要

マルチエージェントシステムは、複数の自律したエージェントが協調的に処理を行うシステムである。各エージェントは、それぞれ並列して動作する分散協調アルゴリズムを実行する。これらのエージェントがメッセージのやり取りを通じてお互いの情報を交換し合い、それらの結果を自身の状態に反映させることで、協調的な処理を行う。この際、一般にこれらのエージェントを集中的に管理するものは存在しない。

2.2 マルチエージェントシステムの多様性

マルチエージェントシステムには様々なモデルがあるが、その中には似たような特徴を持つものがあり、それらをいくつかの種類に分類することができる。本研究では、その中でも比較的よく使われ、汎用性が高いネットワーク型モデルと、グリッド型モデルを対象とした。

2.2.1 ネットワーク型モデル

ネットワーク型モデルは、ノードとリンクからなるグラフ構造によるシステムを表現するモデルである。ネットワーク型モデルでは、個々のエージェントはノードとして表現される。通信可能な相手との関係はリンクとして定義される。これにより、メッセージ交換に基づく基本的なマルチエージェントシステムの多くを表現することができる。

2.2.2 グリッド型モデル

グリッド型モデルでは、一般的に格子（グリッド）状のフィールドを用いる。各エージェントは、グリッド上に配置され、必要に応じて他のグリッドに移動する。また、グリッドの各マスに価値を設定し、その価値に

応じた動作をエージェントに行わせるモデルがある。

3. マルチエージェントシステムにおけるプログラミングの課題

マルチエージェントシステムを実装する際のプログラミングにおける課題を示す。

3.1 従来言語の問題点

一般的によく用いられるプログラミング言語として、C や Java などが挙げられる。しかし、これらの言語は手続き型言語であるため、必ずしも非同期的な分散アルゴリズムの記述に適しているとは言えない。また、ホスト間通信にソケットを使用するため、分散ホスト環境に対応した分散プログラムの記述が煩雑になる場合がある。また、スレッドの管理コストが大きいため、スケーラビリティに問題が生じる場合がある。一方、Erlang はステートレスなプログラミング言語であり、プロセスが非常に軽量であるため、スケーラビリティに優れている。また、純粋な関数型言語であるため、宣言的にプログラムの記述ができる。また、単一ホスト上のプロセス間通信を行う場合と、異なるホスト間でのプロセス間通信を行う場合で同様に、比較的簡素に記述できる。

3.2 Erlang の問題点

Erlang でマルチエージェントシステムを実装する上で問題となるのが、Erlang における状態の非保持性である。実装するマルチエージェントモデルによっては、様々な大域的な状態を扱う必要が生じる。例えば、グリッド型モデルでは、グリッドは大域的に管理する必要がある。Erlang でも大域的な状態を扱うことも不可能ではないが、Erlang のプログラミングの基本的な方針から外れてしまう上に、ユーザの負担も大きくなってしまう。そこで、本研究では、Erlang で状態を容易に管理できるようにするためのフレームワークを構築した。

4. 提案シミュレーションシステムの概要

本研究で実装したマルチエージェントシミュレーションシステムについて説明する。提案システムでは、Erlang で大域的状態を扱えるようにし、状態の非保持性によるユーザーの不都合を解消する。そこで、管理者と呼ばれる特殊なプロセスを導入し、エージェントの管理など、大域的な状態に関わる機能を持たせた。管理者の動作は全てフレームワーク内に隠蔽し、ユーザが管理者を意識する必要性を軽減した。また、後述のように、個々のエージェントに状態を持たせる必要が生じた場合も、なるべくユーザが状態を意識しなくて済むようにした。

4.1 ネットワーク型モデルへの適用

まず、ノードは Erlang のプロセスとして表現する。また、ネットワーク型モデル用シミュレーションシステムを実現するには、隣接ノードなどを用いて、ノード間のリンクを表現する必要がある。そこで、それぞれのプロセスの引数として状態変数を持たせた。状態変数には、エージェントの色、座標、隣接エージェントなど、シミュレーションに応じて必要な情報が含まれる。状態変数は、フレームワークにより管理され，プログラムからはその値の参照、変更を行う。

動作の基本的な流れは次のようになる。まず、初期化処理が行われ、管理者プロセスが作成される。その後、エージェントがフレームワークを通して生成される。この際、エージェントが生成されたことは管理者に報告され、大域的な情報として維持される。

4.2 グリッド型モデルへの適用

グリッド型モデルでは、エージェントの座標などの情報、エージェントが配置されるグリッドを管理することが必要である。そこで、ネットワーク型モデルと同様、エージェントを Erlang のプロセスとして表現し、引数として状態変数を持たせた。また、グリッドの情報は管理者が保持するものとした。

動作に関しては、基本的にネットワーク型モデル用シミュレーションシステムと同様であるが、グリッド型モデルでは隣接ノードのような状態は持たず、エージェント同士で通信する際は、グリッド情報の管理者から任意の座標のエージェント情報を取得する。また、グリッドの情報は全体で一貫している必要があるため、エージェントが移動する際に、管理者の保持する情報を同時に更新する。

4.3 提案システムの分散化

スケーラビリティに優れる Erlang の特長を活用するため、分散ホスト環境上でのマルチエージェントシステムを容易に実装できるようにした。分散ホスト環境のモデルでは、全体の管理者とは別に、ホストごとに副管理者を設け、管理者の負荷を分散した。

4.3.1 ネットワーク型モデルの分散化

ネットワーク型モデルでは、エージェントが生成される際、実行環境として、最も負荷の少ないホストが選択される。エージェントが生成されたことは、全体の管理者ではなく、そのホストの副管理者に報告される。これにより、分散ノード環境でボトルネックとなりうる管理者の影響を削減する。また、ホスト間通信の回数が減少し、システム全体の通信コストが減少する。

4.3.2 グリッド型モデルの分散化

グリッド型モデルでは、全体のフィールドを分割し、各部分フィールドを各ホスト上に配置する。各部分フィールドの情報は、各ホストの副管理者が管理する。また、各エージェントがそれぞれ同時に複数の座標の価値を更新するとき、複数のホストの情報を同時に更新する必要がある。このような場合のグリッドの情報の不整合を避けるため、複数のホストの情報を同時に更新、参照する際は、一定の順番でホストにアクセスするようにした。また、グリッド型モデルでは、自エージェントの座標付近のグリッドの情報を利用することが多い。しかし、エージェントのプロセスは常に生成されたホスト上に存在するため、自エージェン

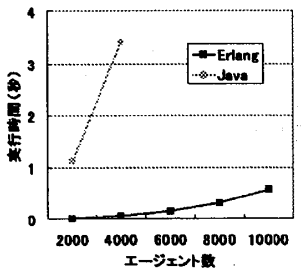

図 1　エージェントの生成時間

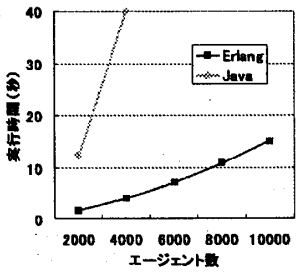

図 2　シミュレーションの実行時間

トの座標付近のグリッドの情報が同一ホスト上にあるとは限らない。そこで、エージェントが別のホストで管理されているグリッドに移動した場合は、エージェントのプロセスそのものをエージェントの移動先座標を管理するホストに移動させる、マイグレーション機構を導入した。

5. 実験と考察

提案手法について、実験による評価と考察を示す。

5.1 Erlang と Java の比較

マルチエージェントシステムの実装における Erlang の有効性を評価・検討するため、Erlang と Java をそれぞれ用いたマルチエージェントシミュレーションプログラムを実装し、比較実験を行った。実験には、グラフの彩色問題を DSA(Distributed Stochastic Algorithm)[2] によって解くプログラムを用い、エージェント（スレッドやプロセス）の生成に掛かる時間と、シミュレーションを 100 サイクル繰り返すのに掛かる時間を測定した。グラフの彩色問題は、隣接エージェント同士で色が重ならないように自エージェントの色を決める問題である。DSA は、確率的に分散制約充足最適化問題を解く手法の一つである。DSA の流れは、次のようになる。まず、各エージェントは自身の現在の変数値を隣接エージェントに送信し、隣接エージェントからそれぞれの変数値を受信する。次に、受信した変数値と自身の現在の変数値における評価値を求める。そして、評価値が最大となる自身の変数値を求め、最大評価値が現在の評価値を上回っていれば、確率 P1 で自身の変数値をその値に変更し、それ以外の場合は確率 P2 でその値に変更する。これを 1 サイクルとする。

実験には、CPU が Intel Pentium 4 2.80GHz、メモリが 1GB の計算機を使用し、Erlang R11B-5 を用いた。パラメータは、P1 = 0.7、P2 = 0.3、色数= 3、各エージェントのリンク数= 4 とし、エージェント数を変化させた。

実験結果を図 1、及び図 2 に示す。エージェントの生成時間、シミュレーションの実行時間共に Erlang が Java を下回っている。エージェント数が増えた場合の実行時間の上昇は、Erlang の方が緩やかであり、Java と比較して Erlang のスケーラビリティの高い結果となった。また、Java の実験結果において、6000 エージェント以上の場合の結果が無いのは、メモリ不足でスレッドが作成できなかったためである。一方、Erlang では単一ホストで少なくとも 1000000 プロセスまで同時に実行できることを確認した。

次に、Erlang と Java の記述性の比較を行うため、それぞれのプログラムのエージェントの動作部分のソースを示す。Erlang を利用して宣言的にプログラムを記述することで、直感的に理解しやすいプログラムになることが確認できた。なお、この例では、Erlang の記述量が Java より増加している。但し、Java の例は、Thread を用いて簡略された単一ホスト上でのみ実行可能なプログラムである。一方、Erlang の例はメッセージ通信を用いていて、分散ホスト上でもほぼ同一のプログラムを使用できる。

Erlang のプログラム

```
sendw(List, Msg) ->
  send(List, Msg),
  recv(List).

eval(Val, List) ->
  lists:sum(lists:map(fun(Elem) -> case Elem of Val -> 0;
    _ -> 1 end end, List)).

max_eval(Val, ValList) ->
  max_eval(Val - 1, ValList, {Val, eval(Val, ValList)}).

max_eval(Val, ValList, {MaxVal, MaxEval}) ->
  case Val of
    0 -> {MaxVal, MaxEval};
    _ ->
      Eval = eval(Val, ValList),
      if
        Eval > MaxEval ->
          max_eval(Val - 1, ValList, {Val, Eval});
        true ->
          max_eval(Val - 1, ValList, {MaxVal, MaxEval})
      end
  end.

select({Val, Eval}, {NewVal, NewEval}) ->
  Rnd = random:uniform(),
  if
    ((NewEval > Eval) and (Rnd < 0.7)) or
      ((NewEval =< Eval) and (Rnd < 0.3)) -> NewVal;
    true -> Val
  end.

agent(Val, NextNode, Cycle) ->
  case Cycle of
    0 -> ok;
    _ ->
      ValList = sendw(NextNode, {self(), MyVal}),
      agent(select({Val, eval(Val, ValList)},
        max_eval(3, ValList)), NextNode, Cycle - 1)
  end.
```

Java のプログラム

```
class Agent extends Thread {
  public void run() {
    for(int cycle = 0; cycle < 100; cycle++) {
      int eval = 0, maxEval = 0, maxVal = 0;
      for(int i = 0; i < 4; i++)
        Dsa.thread[next[i]].msgBox.add(new Msg(id, myVal));
      for(int i = 0; i < 4; i++) {
        Msg msg;
        while((msg = (Msg)msgBox.poll()) == null) yield();
        if((val[i] = msg.msg) != myVal) eval++;
      }
      for(int i = 1; i < 4; i++) {
        int tmpEval = 0;
        for(int j = 0; j < 4; j++) {
          if(val[j] != i) tmpEval++;
        }
        if(tmpEval > maxEval) {
          maxEval = tmpEval;
          maxVal = i;
        }
      }
      double rnd = Math.random();
      if((maxEval > eval && rnd < 0.7) ||
        (maxEval <= eval && rnd < 0.3))
        myVal = maxVal;
    }
  }
}
```

5.2 提案手法を用いた、分散ホスト環境におけるスケーラビリティ

次に、提案システムにおいて、ホスト数を増やして大量のエージェントを実行した場合のスケーラビリティを調査した。実験には、ネットワーク型モデル用シミュレーションシステムでは、グラフの彩色問題を DSTS(Distributed Stochastic Tabu Search)[3] によって解くプログラムを用いた。DSTS は、前の実験の DSA にタブー探索を導入した手法である。

グリッド型モデル用シミュレーションシステムでは、Carry and Drop モデルのプログラムを用いた。Carry and Drop モデルは、次のような動作をするモデルである。まず、グリッド上にお金をいくらかばら撒き、エージェントをいくつか配置する。エージェントは、グリッド上をランダムに移動し、止まったマスにお金があった場合は、それを回収する。他のエージェントがいるマスに移動しようとした場合は移動できず、そのエージェントに自分のお金を 1 渡す。これを 1 サイクルとして、各エージェントに寿命を設定し、全てのエージェントが消滅するまでの時間を測定した。

実験には、CPU が Intel Pentium 4 3.00GHz、メモリが 2GB、LAN がイーサネット（100Mbps）の計算機を使用し、Erlang R11B-5 を用いた。パラメータは、グラフの彩色問題は前の実験と同様で、タブー期間を 1 としている。Carry and Drop モデルでは、グリッドのサイズ＝ 100 × 100、お金の合計＝ 2000、エージェントの寿命＝ 500 とし、エージェント数を変化させた。

実験結果を図 3、及び図 4 に示す。 mig. は、エージェントが別のホストで管理されているグリッドに移動した際に、エージェントのプロセスを移動先のグリッドを管理するホストにマイグレーションしているか否かを示す。

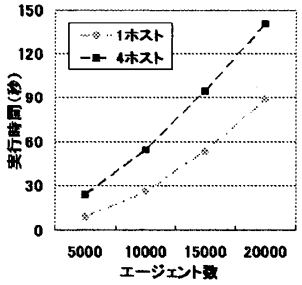

図 3 グラフの彩色問題実行時間

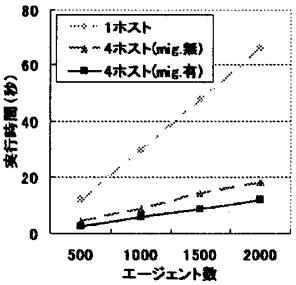

図 4 Carry and Drop 実行時間

グラフの彩色問題では、1 ホストの場合より、4 ホストで分散化した場合の方が、実行時間が増えている。これは、分散化させたことによる各エージェントの処理の効率化と比較して、ホスト間通信のコストが大きかったためと考えられる。

Carry and Drop モデルでは、グリッド情報を管理する管理者がボトルネックになっているため、4 ホストでグリッドを分散して管理するようにしたことで、実行時間が短縮できたものと考えられる。また、マイグレーションを行うことで実行時間が短縮できたのは、ホスト間通信が減少することで、通信コストが削減できたためと考えられる。

6. ま と め

本研究では、Erlang を用いたマルチエージェントシステムの実装の有効性を評価・検討するため、Erlang と Java をそれぞれ用いたマルチエージェントシミュレーションプログラムを実装し、比較実験を行った。その結果、スケーラビリティや記述性の面で、Erlang に有効性があることが確認できた。また、Erlang によるシミュレーションシステム構築の問題点を解決するフレームワークを導入し、分散処理化に対応した。今後の課題としては、実際的なマルチエージェントシステムの実装方法についての検討、及び分散ホスト環境での実装における他言語との比較などが挙げられる。

参 考 文 献

1) Armstrong, J.: Making reliable distributed systems in the presence of hardware errors, PhD Thesis, The Royal Institute of Technology Stockholm, Sweden (2003).
2) Zhang, W., Wang, G. and Wittenburg, L.: Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance, *AAAI-02 Workshop on Probabilistic Approaches in Search* (2002).
3) 飯塚泰樹，鈴木浩之：Tabu Search を用いたマルチエージェント型分散制約充足手法の提案，電子情報通信学会技術研究報告 (2006).