B_021

自律分散処理システムのための強マイグレーション化モバイルエージェント
Design and Implementation of Strong Migration Mobile Agent System for Autonomous Distributed Processing Systems

桜井 康樹*1　田久保 雅俊*1　佐々木 竜介*2　甲斐 宗徳*1
Yasuki Sakurai　Masatoshi Takubo　Ryusuke Sasaki　Munenori Kai

1. はじめに

並列分散処理を行うことにより、ネットワーク上に接続された複数のマシンに処理を分散させ、単一のプロセッサにかかる負荷を軽減させることが可能となり、処理の高速化が期待できる。

一般に分散処理システムでは、システム全体の状況把握や管理が必要となり、プログラム記述者がそれらの制御をすべて含めてアプリケーションプログラムを記述するのは非常に困難である。そのような困難さを解決し、さらに処理の効率化や高信頼性・耐故障性を維持するためには、システム自身の自律性が必要となる。そこで我々はシステムに自律性を持たせ、システム管理者とプログラム記述者の管理負担の軽減、信頼性・耐故障性の向上という目標を満足するため、モバイルエージェント技術を用いた Java ベースの自律分散処理システムを試作してきた。

しかし、それは多くの Java ベースのモバイルエージェントシステムと同様に、弱マイグレーションのモビリティを利用しているため、移動先で移動前の処理を継続するエージェントを自由に記述することは困難である。そこで、利用してきたモバイルエージェントシステムのモビリティを強マイグレーション化して、移動前の処理を再開する記述が可能なエージェントの実現方法を提案する。

2. 従来の自律分散処理システムの概要

2.1 自律分散処理システムとモバイルエージェント

自律分散処理システムとは、システムの状況の変化にシステム自身が自律的に対応しながら分散処理を行うシステムである。一般的に分散システムにおいて、タスクの負荷分散や停止したタスクの再生処理等の作業にはシステム全体の状況把握や管理が必要であり、それらのコントロールをすべて考慮してプログラム記述者がアプリケーションプログラムを記述するのは非常に困難であると考えられる。そこで、あらかじめシステムに自律性を持たせることで、これらの問題を解決することができる。システムの自律性を満たすためには、システム内で動く各ソフトウェアが、他からのメッセージに応じた行動と、自ら取得した外部の情報に応じて判断を行い、その結果によって行動を計画し、その計画に基づき行動できる必要がある。そこで本研究では、自律分散処理システムを実現するにあたって、モバイルエージェントを利用することにした。

エージェントとは、人間の代理としてシステム上で自律的に行動するソフトウェアであり、モバイルエージェントとは、ネットワークを通じてマシン間を移動しながらタスク処理を行うことができるエージェントのことを言う。そして、モバイルエージェントシステムとは、モバイルエージェントの動作（生成、消去、移動、保存、複製、通信など）を提供するシステムのことである。既存のモバイルエージェントとしては、AgentSpace（佐藤一郎氏）[1]や Aglets（日本 IBM 社）[2]、JavaGO（東京大学米澤研究室）[3]、MOBA（首藤一幸氏）[4]等が挙げられる。

2.2 AgentSpace を用いた自律分散処理システムの概要

我々の自律分散処理システムの目的は、「分散処理の手法について詳しくない人でも簡単に分散処理の恩恵を受けられるシステムを構築すること」と、「様々な種類のプログラムを実行可能な汎用性の高いシステムを構築すること」である。この２つの目的のために、モバイルエージェントを利用することにした。既存のモバイルエージェントを選択する上で、本研究では、モバイルエージェントシステムのソースが公開され改造が可能となっていて、エージェント移動時に情報を圧縮して送るため移動が速いという理由から、AgentSpace を用いて自律分散処理システムのプラットフォームの開発を行っていた。

1 つ目の目的を満たす為に、分散処理の開始やシステムの稼動状態の把握が容易となる様に GUI を用いて操作ができるようにした。また、2 つ目の目的である様々なプログラムを容易に書けるようにするために、SPMD(Single Program Multiple Data)形式のプログラムを記述できるようにした[5]。

図 1 は、自律分散処理システムの構成の一例を示したものである。

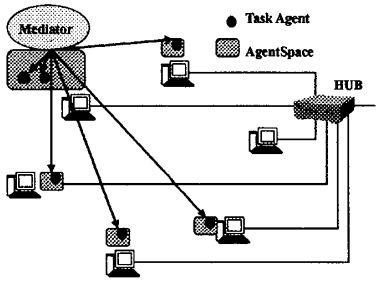

図 1　自律分散処理システム構成例

図 1 に示すように、システムは LAN の様なネットワークに接続された複数のマシンからなる。エージェントは AgentSpace 上にしか存在できないので、分散処理に参加するコンピュータでは AgentSpace を起動させておく必要がある。Mediator はユーザとのインターフェースとなるエージェントで、ユーザから実行するタスクを記述したプログラムファイルなどの必要な情報を入力されると、その情報

*1 成蹊大学大学院情報処理専攻
*2 横河電機

に従いそのタスクを分散処理するタスクエージェントを生成する。

タスクエージェントはMediatorによってうまく負荷分散されるようにネットワーク上のコンピュータに振り分けられて移動する。それぞれのコンピュータに移動後、タスクエージェントはプログラムを実行して解を求める。解を求めたタスクエージェントはMediatorのあるコンピュータに戻り、Mediatorに解を伝えて消滅する。Mediatorは、タスクエージェントから受け取った解を統合し、それをユーザに表示する。

ユーザが要求したタスクを実質的に分散処理するタスクエージェント以外に、自律的に処理を行うために様々な支援を行う以下のようなエージェントを投入している。

①管理エージェント[6]

システム内を巡回しながら、システム全体の動的な情報を取得、更新する。その情報の中から、AgentSpaceがダウンするなどの原因でタスクエージェントが消えてしまったことを検出でき、新しく代わりのタスクエージェントを生成し、処理性能の一番高いコンピュータに再配置することができる。これにより、ユーザが要求した処理が完了できない事態に陥らないようにできる。

②分散管理モニタエージェント[7]

現在システムがどのように動いているのか、といったシステムの振る舞いを表示するためのエージェントである。管理者が人為的にエージェントを操作しシステム制御を行いたい時に必要となる情報を管理者に伝える目的で導入した。この分散管理モニタにより、システム状態の表示に加えて、エージェントの生成・派遣・消去なども行える。

③MediatorSupporterエージェント[8]

Mediatorの生成と同時に生成され、各AgentSpace間を巡回しながら、生成されたMediatorとそのMediatorの存在しているAgentSpaceを監視する。MediatorまたはAgentSpaceの異常終了を確認したら、各タスクエージェントの監視を始め、結果の求まったタスクエージェントから結果を収集し結果がすべて集まったら他のMediatorにその結果を出力できる。

④エージェント間通信用エージェント[9]

AgentSpaceが持っていた、同じAgentSpace内でのエージェント間通信機能に加え、別AgentSpace間でのエージェント間通信を行えるようにしたエージェントである。これは通信要求のあるエージェントから通信データを受け取り、宛先エージェントを探して届ける。

これらのエージェントの他に、分散処理エディタというアプリケーションも実装している。これは分散処理用アプリケーションを記述する時に、通信命令を記述しやすいようにサポートし、作成後も、通信命令がどのソースの、どのメソッドと通信しているかが簡単に見て取れる機能を提供する。

2.3 強マイグレーションの必要性

ネットワークに繋がれたコンピュータの性能は全て同じではなく、各コンピュータは分散処理以外の処理も行っているため発揮できる性能は常に一定ではない。そこで本システムでは、分散処理開始時直近における各コンピュータの性能値に合わせて自律的に各コンピュータが行う仕事量を変化させる機能を構築している。

分散処理を開始するときにMediatorによって生成されるタスクエージェントは、SPMD形式のプログラムとして記述されているため同じ仕事量を持っていて、このタスクエージェントの数が各コンピュータに割り当てられる仕事量に相当する。タスクエージェントを生成したMediatorは性能値の高いコンピュータにより多くのタスクエージェントを割り当てていく。

ただしこの割り当ては、分散処理開始時の各コンピュータの性能によるものであり、一度分散されたタスクエージェントがその後の状況に応じて自由に移動できるわけではない。例えば、分散処理中のコンピュータに分散処理とは別の要因で非常に大きな負荷がかかり、タスクエージェントの処理の完了が大幅に延期されるというような事態においてもエージェントはただ待つしかない。そこで、そのような場合にシステムが自動的に性能の良いコンピュータへタスクエージェントを移動させ、大幅な処理の遅延を防ぐ機能が必要となる。

この機能実現には、エージェントが任意の中断した時点の実行時データを保持し、移動先のコンピュータ上で中断時点からの処理の再開ができることが必要となる。

しかし、現在利用しているモバイルエージェントシステムAgentSpaceのモビリティは弱マイグレーションであるため移動時に持っていく実行時データでは、中断した時点からの再開を行うには不十分である。そこで、本研究では、中断時点での再開が可能な実行時データを持って移動する強マイグレーション方式のモバイルエージェントシステムを実装することにした。

3．エージェントの強マイグレーション化

3.1 Javaによるモバイルエージェントの問題点

Javaにおいてプログラムが使用するメモリ領域には、実行コード領域（プログラムの実行コードを格納する領域）、スタック領域、ヒープ領域がある。

しかし、Javaを用いてモバイルエージェントを実装する際に、予めJavaに備わっているシリアライズ機能を用いることによって、実行コードに加えてヒープ領域内の情報の保存が可能となっているが、スタック領域内の情報やプログラムカウンタを実行状態として保存する機能は現状のJavaVMではサポートされておらず、これをJavaで実装するのは容易ではない。そこで、Javaを用いた既存の強マイグレーション方式のモバイルエージェントは、これを満たすためにJavaVMの変更かソースコード変換を行っている。JavaVMを変更する場合、JavaVMのバージョンアップごとに修正を行わなければならず、システム開発の負担が大きい。そのため、Javaを用いたモバイルエージェントシステムの多くが、シリアライズ機能だけを用いて実装された、完全に実行状態の保存されていない弱マイグレーション方式のエージェントを採用していた。

3.2 強マイグレーション化のための手法

本研究では、JavaVM に手を加えることなく強マイグレーション方式のモバイルエージェントシステムの開発を行えるように、以下に述べる方法でスタック領域の情報の取得とソースコード変換を行った。

3.2.1 スタック領域の取得

スタックの中身を取得するために、JPDA(Java Platform Debugger Architecture)を使用した。これは、サン・マイクロシステムズ社が提供する JavaVM に標準的に実装されており、Java アプリケーションのデバッグに使用されるもので、実行中のスレッドや実行情報へのアクセスが可能となっている[10]。本研究では、この機能を用いることによって、スタック領域のデータを取得することにする。

エージェントは、スレッド上で実行される。そこで、エージェントの移動の際に JPDA を用いてエージェントが実行されているスレッドにアクセスし、スタック内の情報を取得し、シリアライズ可能な変数に格納する。そして、実行コードやヒープ領域内のデータと共にシリアライズして、移動できるようにする。

3.2.2 ソースコード変換

実行状態を完全な形で保存するためには、スタックの中身だけでなく、プログラムカウンタも必要となってくる。JPDA では、このプログラムカウンタに類似したデータを扱っており、これを取得することは可能となっている。しかし、この取得したデータを用いてプログラムを途中から実行する機能は、JPDA でもサポートされていない。そこで、本研究ではソースコードの変換を行うことにより、同等の機能をサポートすることにした。

ソースコード変換は、図 2 のようにユーザコードをソースコード変換器にかけることによって、ステートメントコード、ランタイムコード、ステートメントフロー情報を取得することにより行われる。

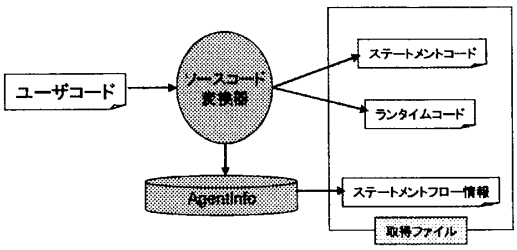

図 2　ソースコード変換

①ステートメントコード

ユーザコードの構造を解析し、ユーザコードから提供されるプリミティブによって分割したステートメントを statementN（N＝整数）というメソッドに、if の内部（条件部）のステートメントは branchStatementN というメソッドに書き込んだコードのことである。また、ソースコードを解析して繰り返しの回数がわかる for 文に関しては、現時点においてはその for 文をアンローリングすることによって、ステートメントコードを生成する。ソースコードを解析しても繰り返し回数がわからない for 文や、while 文、再帰関数に関しては、現時点においてソースコード変換器でのステートメントコードへの変換には、未対応である。

図 3 は、ユーザコードのステートメントコードへの変換の例を示している。

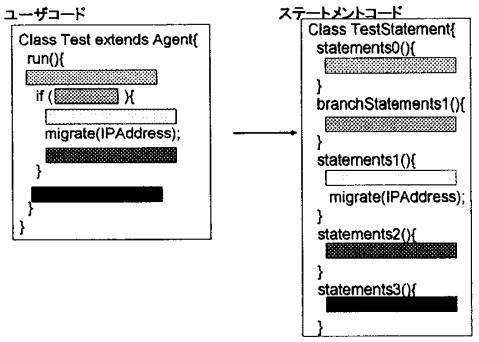

図 3　ユーザコードとステートメントコード

また、ユーザコードに対して提供されるプリミティブには、次の 5 つがある。

- エージェントの生成：create(fileName)
- エージェントの消去：destroy()
- エージェントの複製：duplicate()
- エージェントの移動：migrate(IPaddress)
- エージェントの保存：backup()

図 3 のように、ステートメントコードは細かくステートメントごとのメソッドに分割されている。弱マイグレーション方式のモバイルエージェントシステムでは、移動後におけるメソッド単位での継続実行が可能である。そこで、本システムでは、移動後にこのステートメントコードのどのステートメントから処理を行うかを指定することにより、プログラムカウンタと同等の機能を実現している。

②ランタイムコード

ソースコード変換によって得られたステートメントフローの各ステートメントの情報に基づいて、エージェントの動作を管理するコードである。

ランタイムコードの自動生成では、テンプレートを用意し、それを利用して必要な部分を変更しながらランタイムコードを作成するという手段をとった。以下の図 4 はその変換の一部の例である。

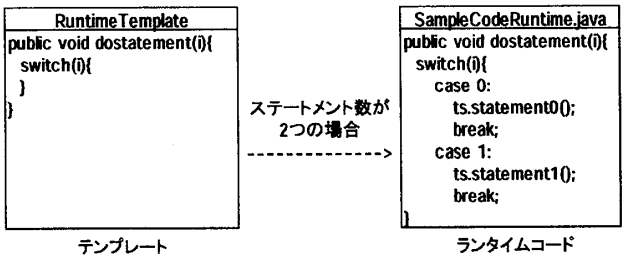

図 4 ランタイムコードの自動生成

dostatement 関数は、ステートメントコードに記述されているコードを実行させるための関数であり、ソースコード変換を行った際に生成されたステートメント数に合わせて switch 文の各 case 文の記述を自動的に決定する。

③ステートメントフロー情報

AgentInfo は、エージェントの実行状態を表すクラスで、

ある実行状態のスナップショットがそのインスタンスとなる。実行状態のデータとは、実行時にロードする.class ファイルや生成したステートメントコードを元にした実行開始ステートメント、現在のステートメント、次に実行するステートメント等の情報である。これは実行に合わせて動的に変化していく。

3.2.3 強マイグレーション化エージェントの移動

ステートメントコードはプログラムカウンタに従った実行と同等の形態を持つ。ランタイムコードは AgentInfo から実行するステートメント番号と、そのステートメントに関する情報を受け取り、取得した情報に対応するステートメントコードのメソッドを JavaVM に実行させる役割を持っている。つまり、ソースコード変換で生成した 3 つのファイルによってユーザコードを実行している。従って、他のコンピュータへマイグレーションするには、3 つのファイルを全て送らないと移動した先で継続実行することができない。そこで強マイグレーション化エージェントは、ステートメントコード、ランタイムコードおよびステートメントフロー情報の 3 つを直列化して他のコンピュータへマイグレーションすることにより、中断された実行状態からの再開を可能とする。

4. 強マイグレーション方式のモバイルエージェントの動作確認

強マイグレーションモバイルエージェントを動作させるシステムを、本研究では AgentSphere と呼んでいる。すなわち、AgentSpace と同様に、AgentSphere を各コンピュータで起動することにより、エージェントをそれらのコンピュータに送り込むことができる。本システムの動作確認として、バブルソートを行うエージェントを 2 台のコンピュータ間を何回か往復させることにした。実行の流れは、対象のプログラムファイルをソースコード変換して 3.2.2 で述べた 3 つのファイルを生成し、その後実行を開始する。このプログラムでは、ソートを行う配列のデータの中で一番小さい値が確定したら次のコンピュータに移動し、次に小さい値が確定したら、再び最初のコンピュータに戻って次に小さい値を確定するという処理を繰り返していく。その結果、バブルソートが正常に完了し、強マイグレーションが正しく動作していることを確認した。

5. 終わりに

スタック領域の取得、ソースコード変換を行うことによって、既存の JavaVM に変更を加えることなく、強マイグレーションエージェントの記述ができるようになった。しかし、移動時に持っていく実行時データが増えるため、通信オーバヘッドが増えることとなる。今後は、このオーバヘッドに関する評価を行って性能向上を目指す。

そして自律分散処理システムへの適用を行っていく。その際、2.2 で述べた各種のエージェントの役割とオーバヘッドの低減を考慮して、適切なモビリティを選択できるように強マイグレーション及び弱マイグレーションのモバイルエージェントを共に利用していく。

参考文献

[1] 佐藤一郎：「AgentSpace：モバイルエージェントシステム」,日本ソフトウェア科学会, Dec.1998

[2] 日本 IBM 東京基礎研究所：http://www.trl.ibm.com/aglets/ , Jul.2006 参照可

[3] 米澤明憲,関口龍郎,橘本政朋：「移動コード技術に基づくモバイルソフトウェア」http://homepage.mac.com/t.sekiguchi/javago/index-j.html, Jul.2006 参照可

[4] 首藤一幸：http://www.shudo.net/moba/ ,Jul.2006 参照可

[5] Ryusuke Sasaki, et al.: "IMPLEMENTATION AND EVALUATION AUTONOMIC DISTRIBUTED PROCESSING SYSTEM USING MOBILE AGENT", Proc.of IEEE Pacific Rim conference, No.237, Aug.2005

[6] 坂井功,相河寛典：「モバイルエージェントを用いた自律分散処理システムの構築 －自律分散処理をサポートする管理エージェントの作成－」2003 年度電気学会電子・情報・システム部門大会講演論文集, pp.1142-1146,Aug.2003

[7] 小川大介,佐治大介,村本洋明：「モバイルエージェントを用いた自律分散処理システムの構築 －タスクエージェントの記述法とタスク分散の効率化－」2003 年度電気学会電子・情報・システム部門大会講演論文集, pp.1116-1120,Aug.2003

[8] 佐々木竜介,遠藤航,青山迅,清水敏宏,小川大介,坂井功：「モバイルエージェントを用いた自律分散処理システムの構築 －自律分散処理システムにおける性能とユーザビリティの向上－」2004 年度電気学会電子・情報・システム部門大会講演論文集, pp.1036-1040,Sep.2004

[9] 田久保雅俊,佐々木竜介,内藤智史,小川大介：「モバイルエージェントを用いた自律分散処理システムの構築 －エージェント間通信命令の実装と分散処理エディタの試作－」2005 年度電気学会 電子・情報・システム部門大会講演論文集, GS15-4, pp.1034-1039,Sep.2005

[10] Torsten Illmann,et.al, "Transparent Migration of Mobile Agents Using the Java Platform Debugger Architecture", Mobile Agents: Proceedings of the 5th International Conference, Ma 2001 Atlanta, Ga, December 2-4, pp.198-212, 2001