B-003

強マイグレーションモバイルエージェントシステム AgentSphere における エージェントの活動管理

Management of Agents' Action in the Strong Migration Mobile Agent System AgentSphere

山口 大祐 †　市川 顕 †　白谷 浩次郎 †　甲斐 宗徳 †
Daisuke Yamaguchi　Ken Ichikawa　Kojirou Hakuya　Munenori Kai

1. はじめに

著者らのモバイルエージェントシステムは Java で実装されており、各エージェントがマシンやネットワークの負荷状況を考慮して移動を行うことで、自律的に分散処理を実現しようとするものである。また、エージェントのモビリティには強マイグレーションを採用した。強マイグレーションを実現するシステムとしては JavaGo[1]などの先例があるが、これらは Java 仮想マシン(JVM)を独自拡張することによって実現されている。本研究では、JVM を変更せずに強マイグレーションを実現するアプローチをとり、ユーザが記述した強マイグレーションのコードを通常の JVM で実行できるように、独自の自動コード変換器を開発し、またそのエージェントが活動する空間 AgentSphere を開発してきた [2][3]。本論文では、エージェントの活動を管理する機構についての設計と実装を報告する。機能としては、複数エージェントの一括ロード機能と負荷分散のためのスケジューラ機能である。

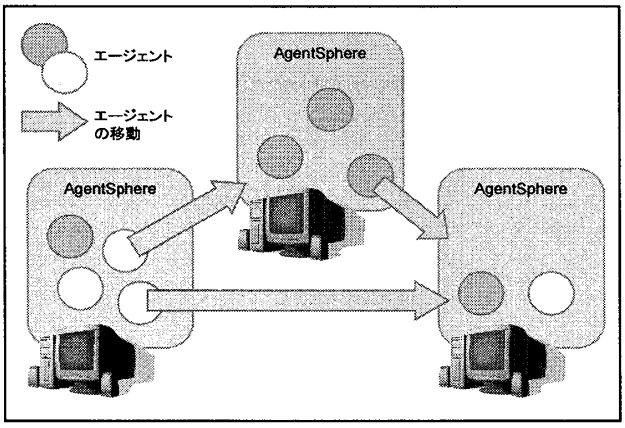

図 1-1：AgentSphere 概要図

図 1-1 は本研究で開発している AgentSphere の概要である。AgentSphere を起動しているマシン上で実行されているエージェントが、実行環境の負荷変動などを検出し、自律的にマシン間を移動する様子を示している。また、これらのエージェントは互いにメッセージ機構と呼ばれるエージェント間通信機能を用いて協調動作を行うことが可能になっている。

このエージェントの協調活動を支援するため、エージェントの一括ロード機能は複数エージェントをまとめて一つのグループに設定した上で起動することにより、協調相手の存在を保証する機構である。これに伴い、協調動作の失敗などを検知できるようエージェントの実行監視機構も実装した。

また効率的な負荷分散のためにはエージェントが活動するマシンを選定する必要がある。そのためスケジューラ機能として、分散処理に参加するマシンの性能値を定期的に更新する機構を作成し、それに基づくエージェントの負荷分散手法についても考察した。

† 成蹊大学理工学研究科理工学専攻　Graduate School of Science and Technology, Seikei University

2. 一括ロード機能

エージェントの協調相手の存在を保証する機能として、一括ロード機能(GroupLoader クラス)を開発した。この GroupLoader は、起動するエージェントのクラスを逐次受け取り、その後実行許可を受け取る。実行許可が来た段階でそれまでに受け取ったすべてのエージェントの実行を開始させる。

2.1 エージェント間通信

複数のエージェントを用いて効率良く仕事をするためには協調した動作を行う必要がある。そのための方法としてメッセージ機構[2]を開発してきた。この機構の通信方法として Send(非同期送信),call(同期送受信),future(非同期送受信)を用意した。

2.2 エージェントの一括起動

エージェント間通信を行うには、通信対象となるエージェントがロードされていなければ通信することができない。そのため、各エージェントにとって協調相手のロードがすでに行われていることが保証されている必要がある。通信相手が既にロードされていることを保証するために、一つの仕事を行うエージェント群を一つのグループとして纏めるとともに、一括起動できるようにする必要がある。

エージェントの記述方法は、初期処理（create メソッド）、移動するエージェントの移動後の処理（arrive メソッド）、終了処理（destroy メソッド）を記述し、実行も同様の流れで行われる。

移動しないエージェントでは create メソッド内に終了処理以外すべての処理、すなわちデータ通信の必要があればそれも含めて記述する事になっている。そのため、create メソッド実行の段階から協調動作が可能になっている必要がある。

エージェントの実行は、ユーザがエージェントの基底クラスのメソッドをオーバライドしたメソッド実行である。そのためユーザビリティ向上のためには、メソッド内で単一起動のエージェントとグループ起動のエージェントで区別なく内部処理を書けるようにしたい。

実現手法として図 2-1 で示されているとおりである。

グループ機構を利用する側は、まずグループとして設定したいエージェント群全エージェントのロード要求を GroupLoader へ逐次送る。その後、GroupLoader へ実行許可を出す。GroupLoader では渡された要求に基づいてロードとスレッド割り当てを行う。LowLevelScheduler を介して全スレッド割り当てが終了した事を保証し、実行許可が来た段階で、各エージェントへ create メソッド実行の許可を出すブロッキング機構を追加する方法を選択した。

結果として、create メソッドの段階から自由にメッセージ通信を行えるようになった。

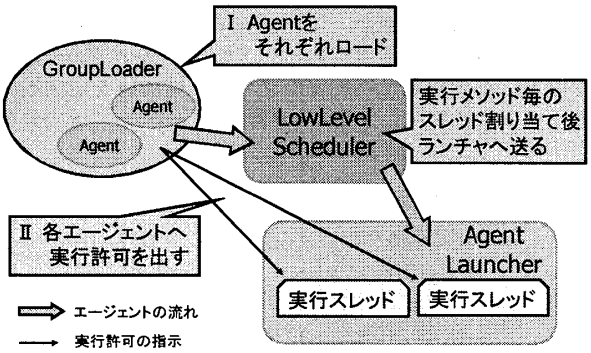

図 2-1：並列起動の実現方法

2.3 禁止動作

エージェントのロードはスレッドのインスタンスを生成しているのと似ていると考え、一つのエージェント群のスタートは 1 度に制限した。これは、Java のスレッドがひとつのインスタンスでスタートメソッドを複数回呼べないようになっていることを参考にしたものである。

3. エージェントの実行監視

エージェントの実行中に停止したままになるなど、エージェントの異常状態を自動検知し、その状態をユーザに通知する機構として「エージェントの実行状態監視」を開発した。この機構は、各マシンで実行中の各エージェントの状態を定期的に確認する。このとき長時間停止状態のエージェントを見つけた場合にはその旨をユーザに通知する。また、メッセージ機能の返信機能を用いて停止している場合には、対象エージェントを探索する。

3.1 エージェントの実行状態の取得

各エージェントは各マシンで実行用スレッドが割り当てられそのスレッド上でメソッドが実行されることにより実行される。このため、エージェントの実行状態とはスレッドの実行状態であり、Java のスレッドには Java 定義で次の状態がある。

- 起動前 : NEW
- 動作中 : RUNNABLE
- 割り込み待機 : WAITING
- 時間制限付割り込み待機 : TIMED_WAITING
- モニタロックを待機 : BLOCKED
- 終了後 : TERMINATED

また、スケジューリングなどに利用できるエージェントの CPU 使用率を得るために、対象スレッドに与えられた総 CPU 使用時間（nsec.）を取得することができる。

これらを用いてエージェントの実行状態を表現すると次のようなものになる。スレッドの実行状態を確認するため、定期的（インターバル 1 秒）にスレッドの状態を確認する。そこで得られたデータを元に、スレッドステートカウンタと、CPU 使用率を用意してエージェントの実行状態を表現する。

スレッドステートカウンタとは、定期確認をする中でスレッドの同一状態の連続を数えるものである。このとき、CPU 使用時間が進んでいたときはこのカウンタをリセットする。理由は、スレッドの状態は動作中以外の状態は全てスレッドが待機している状態なので、計測の合間に状態に変化があったことを検知するには、CPU 使用時間が更新（増加）されていることを確認し、増加していたら同一状態から脱したと判断できるからである。

CPU 使用率は、インターバル時間に割り当てられた CPU 使用時間をインターバル時間で割ったものとする。

3.2 実行監視と自動検知手法

3.1 で述べた方法で取得したエージェントの実行状態を用いて、エージェントの異常状態を自動検知する機構を設計した。内容は、データの更新に連動してエージェントの状態の一覧へアクセスする。このとき異常の疑いがある場合にはその旨を通知する。またこのときメッセージの返信待ちにおいて長時間停止している場合には、返信の対象が存在しているかネットワーク上の各マシンで確認する。ネットワーク上に対象エージェントが存在しない場合には、不慮の事故によりエージェントの実行が失敗していると考えられその旨もまた通知する。

これらの実装は図 3-1 のようになる。まず、異常状態について記述されたエージェント(AgentWatcher)がエージェントの状態が格納された Agent テーブルへアクセスする。AgentWatcher で見つかった異常状態の Agent の情報は、その Agent のユーザのマシンへエラー表示エージェントを送ることによって伝えられる。具体的には、エラー表示エージェントが対象マシンへ移動後「通知理由、Agent の ID、スタックトレース」を Java のコンソールに標準エラー出力する。このときメッセージの返信待ちであった場合には対象エージェントをネットワーク上の他マシンでの存在を確認しに行くエージェントを発行し見つからなかった場合にはその旨もまた標準エラー出力することになる。

このように異常状態の検知についてエージェントで表現した。この利点は、今後異常状態定義の追加や再定義した

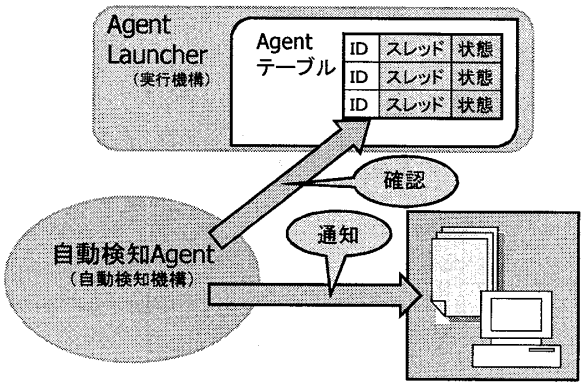

図 3-1：実行監視の実現方法

時 AgentSpehere 本体への改編をせずにサブシステムとして実現できる点である。

4. スケジューラ

スケジューラとは、エージェントの移動先を選定するモジュールである。

自律分散処理システム・AgentSphere を実現するにあたって、ある特定の計算資源に処理が集中し、処理速度の低下、または、メモリ過負荷のためにマシンがダウンするようなことが発生すると、効率的な分散とは言えない。効率的な分散を実現させるためには、AgentSphere が起動し接続されているマシンに、それぞれの性能に合った数のエージェントを分散させなければならない。また、負荷の偏りが発生した場合は、負荷の大きいマシンから負荷の少ないマシンに、エージェントを移動させ、システム内の負荷を均等な状態に保たなければならない。スケジューラが、各マシンの性能値情報を管理するとともに、負荷状況を監視し、負荷状況を考慮した上で、移動のための助言を、自律的に行動するエージェントに行うことで、自律分散を実現する。

4.1 性能値

性能値とは、各マシンの現在の実行性能を表す値のことを指す。この実行性能を計る方法として、CPU 性能値と仮想メモリ量を考慮することにした。また、効率的な負荷分散のためにはこの値を保持管理すると共に、エージェントの再配置やユーザアプリケーションの起動などによって刻一刻と変化する性能値の定期的な更新が必要になる。

4.1.1 性能値の選定

各マシンの実行性能を計る方法として、各マシンの処理速度を計る CPU 性能値と、各マシンのエージェントの許容量を計る仮想メモリ量というものを考えた。

① CPU 性能値

0.1 秒間の整数インクリメントカウント数に変わる新たな性能値として、はじめに、マシンごとに基準となる値を算出し、その値とエージェントの CPU 使用率からマシン間で比較ができる絶対的性能値を算出する。CPU 性能値の算出方法を以下に示す。

システム起動時に 0.5 秒間の整数インクリメントカウント数を測定し、これを A とおく。測定に割り当てられた CPU 時間を測定し、これを B とおく。すると、A/B=1 ナノ秒あたりのインクリメント数となり、これを C とおく。

マシンで稼働しているエージェントの CPU 使用率を平均した値をマシンの CPU 使用率として算出し、これを D とおく。そして、CPU 性能値は C×D で求めることができる。

1 ナノ秒あたりのインクリメント数を算出することにより、マシンごとの性能に依存する値を得ることができる。測定時間の選定法については、測定時間を長くするほど精度が上がるが、誤差を数%以下に抑え、体感的に、気にならない測定時間として、0.5 秒に設定している。また、0.5 秒であれば測定時に負荷をかけた場合も、誤差の少ない値を示すことを実験により確認した。CPU 性能値を使用することで、0.1 秒間整数インクリメントより高い精度の値を得ることができる。また、従来は性能値を更新する度に、0.1 秒間のオーバーヘッドが生じていたが、CPU 性能値の場合は、0.5 秒間の整数インクリメントカウントはシステム起動時の 1 度のみしか行わないため、性能値測定のためのオーバーヘッドを減少させることができる。

②仮想メモリ量

java.lang.Runtime から仮想マシン内の空きメモリ量、メモリ総割り当て量、使用できる最大メモリを、取得する。これらの値から実際に、使用できる仮想メモリの残量を算出する。仮想メモリ量の算出方法を以下に示す。

・割り当て総量－空き量メモリ＝現在の使用量
・最大メモリ量－現在の使用量＝メモリ残量

メモリ残量を取得することにより、メモリ過負荷によりマシンがダウンする前に、エージェントを移動させ、マシンダウンを防ぐことが出来るようになる。

4.1.2 性能値の管理

性能値を管理するための機能として PerformanceInfo クラスを作成した。また、各マシンを識別するためのクラスとして MachineInfo クラスを昨年度までで用意している。

PerformanceInfo は PerformanceMap により、MachineInfo と 1 対 1 で対応させており、MachineInfo をキーとして呼び出しを行う。また、このデータは LowLevelScheduler が保持している。情報の更新などの操作はメソッド群 ToolKit を通して行う。今後、新たに別の値を性能値の指標に加える場合も、性能値情報だけをまとめて管理していることで、作業が容易になると考えられる。また、MachineInfo はブロードキャストを行うデータのため、性能値を別途管理することにより、移動するデータの軽量化を図っている。スケジューラとデータの関係を表した図を図 4-1 に示す。

4.1.3 性能値の更新

各マシンから性能値を更新するエージェントを発行することにより、性能値の更新を行う。

まず、新規に、AgentSphere システムを起動した際に、性能値更新エージェント PerformanceUpdater が、発行元マシンの全ての性能値情報を取得し、AgentSphere が起動し接続されているマシンのリストである MachineList から取得したリストに沿ってマシンを巡回する。そして、AgentSphere が起動し接続されている各マシンにおいて、performanceMap 内の発行元マシンの性能値を更新していく。全てのマシンの巡回が終了した時点で、消滅する。また、新規マシンから性能値更新エージェントを受け取った既存のマシンは自マシンの性能値情報を新規マシンに返信するエージェントを発行する。このようにして、AgentSphere が起動し接続されている全てのマシンが、各マシンの性能値情報を把握することができる。ここまでが、スケジューラに関しての初期動作である。

システムの稼働後は、Scheduler が、定期的に、自マシンの性能値を更新、監視する。前回、性能値更新エージェントを発行してから、性能値に変化が現れた時のみ、再度、性能値更新エージェントを発行し、変化のある性能値情報を各マシンに伝達する。既存マシンからの性能値更新エージェントに対しての返信は行わない。

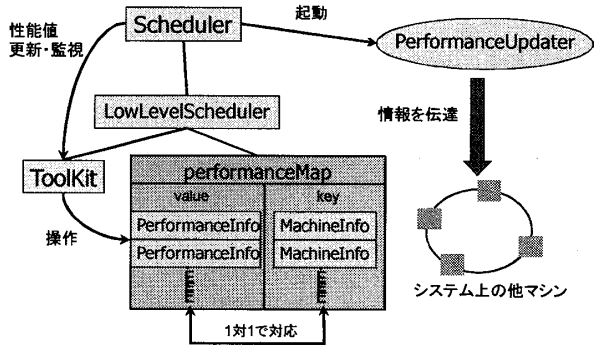

図 4-1: スケジューラ設計図

4.2 エージェントの分散手法

今回、一括ロード機能が実装されたことによりエージェントが複数非常に短い間隔で発行され、性能値の更新間隔より短い間隔でエージェントが複数送り出されることが予想される。この場合、最高性能のマシンにエージェントが集中してしまう可能性がある。これを解決するため新たなエージェントの分散手法を実装した。

全てのマシンの性能値の合計を 100%とし、各マシンがそのうち何割を占めるかを算出し、割合に合った数のエージェントを送り出す。具体的な実装方法を以下に示す。

1．$\frac{\text{マシンの性能値}}{\text{全マシンの性能値の合計}}$＝性能値比率・・・・A

2．A×システム内で稼働しているエージェント数・・B

3．B－それぞれのマシンで稼働中のエージェント数・・・C

4．C の値を比較し、最も大きな値のマシンを移動先として選定する。

エージェントの送り先マシンから、性能値更新エージェントが届く前に、次のエージェントの発行を試みたとしても、送り出したエージェントも考慮して移動先を選定するため、エージェントを同一マシンに送り続けることはなくなる。

これらの分散手法を当てはめるのは、エージェントの移動時である。エージェントの移動には 2 種類ある。

移動できるエージェントは、実際の移動先々で実行される内容の記述された Arrive メソッドの前にその内容を最初に実行するマシンを選択することになる。これを初期分散という。

次に、エージェントは初期分散により性能値に応じて各マシンに配布されるが、エージェントの処理量や実行する環境によって、各マシンで、負荷の偏りが生じる。その際に、負荷の多いマシンから軽いマシンへエージェントを移動させ、AgentSphere 内の負荷を均等にする必要がある。これを再分散という。エージェントが行先を指定せずに移動要求(migrate メソッド)を呼び出した際に、スケジューラはエージェントに保持する性能値情報を元にエージェントが移動すべきかを判定する。負荷が高く、移動を必要とする場合には移動を行う。

これらどちらの場合においても、先の分散手法を用いて移動先を選定することで、AgentSphere 内の負荷状況を均等にし、効率の良い分散処理を行うことができる。また、メモリ過負荷でマシンがダウンする前に、エージェントを他のマシンに退避させることができる。

5. おわりに

本論文では、強マイグレーションモバイルエージェントシステム AgentSphere におけるエージェントの活動を管理する機構についての設計と実装を報告した。活動を管理する機構として具体的には次のことを行った。

エージェントの一括ロード機構というエージェントが安全にエージェント間通信であるメッセージ機構を利用できるようにするための機構を実装した。

エージェントの活動監視という、エージェントの実行中にエージェント実行中の不具合を検知する方法を実装した。

各マシンの実行性能を計る値として性能値の選定と共に、性能値に基づいて AgentSphere 内の負荷状況を均等にし、効率の良い分散処理を行う負荷分散手法の考察を行った。

謝辞

本研究は科研費（基盤研究(C)21500041）の助成を受けたものであることをここに記し、謝意を表します。

参考文献

[1]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.2009 参照可

[2]近藤敬宏・加藤史彬・甲斐宗徳「強マイグレーションモバイルエージェントの自己バックアップ機能とエージェント間通信の実装」,FIT2009, B-010, Sep.2009

[3]赤井雄樹・横内　貴・若尾一晃・甲斐宗徳「強マイグレーションモバイルエージェントシステム AgentSphere の開発」,FIT2009, B-011, Sep.2009