B-006

AgentSphere のためのセキュリティ機構の導入と並列処理能力の評価

Implementation of a Security Feature and Evaluation of Parallel Computing Performance of Mobile Agent System AgentSphere

ディダ ベサリ† 蓮見 建太† 甲斐 宗徳†

Besar Dida Kenta Hasumi Munenori Kai

1. はじめに

本研究で開発されているモバイルエージェントシステムである AgentSphere は自律並列処理を行うプログラム（エージェント）を実行するための環境である。使用言語は Java であり、仮想マシン上で動くことにより Java の環境が整っている限りマシンや OS などに依存しないものである。

開発目的は並列分散処理についての専門的な知識を持たない人でも AgentSphere の環境を利用することで容易に並列分散処理の恩恵を得るためである。現在ではモバイルエージェントシステムの特徴である弱・強マイグレーションに対応しており、実行状態のバックアップ等の機能も備わっているため高い信頼性を有している。

しかし、今までの環境は主に開発を主体としており AgentSphere に行き来するエージェントはすべて善良であると仮定されていたためセキュリティ対策は皆無だった。エージェントの悪利用による成りすまし、データの改ざんや盗聴を防止するためには基本的なセキュリティ対策が必要である。また、現在の AgentSphere は実用的な機能を数多く有しているにも関わらず実際のプログラムによるテストがあまり行われていない。

本論文では新たに実装された AgentSphere 用の権限を設定する SecurityManager の報告、エージェントのアクセス権限によって使用できなくなったファイルアクセス機能の代替として開発されたファイルシステムの報告、そして、既存のマイグレーション機能を使用した AgentSphere の性能評価を報告する。

2. AgentSphere のセキュリティ保護

今までの AgentSphere では自律プログラムであるエージェントの移動先が指定されると、受け入れ先は受け入れざるを得ない構造だった。エージェントは AgentSphere 上であれば受け入れ先の権限を使用して自由にコードを実行することができる。この状態だと不正にデータが改ざん・利用される恐れがあるのでエージェントの権限を設定・管理する SecurityManager を実装した。

2.1 SecurityManager

SecurityManager は AgentSphere の環境で実行され、Policy クラスでエージェントだけではなく AgentSphere 本体にも権限の設定を行うことができる。これを利用することで外部のエージェントや内部の特定のエージェント等の権限を自在に設定することでファイル・システムのプロパティやファイルの操作権を失わせたり与えたりできる。

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

2.2 受け入れ制限

AgentSphere では SecurityManager によって権限が設定できるようになっても外部あるいは信頼していないところからの受け入れ自体を制限する必要がある。よって新たに AgentSphere 同士のエージェントの受け入れを制限する機構が実装された。この機構の必要性は次の二通りである：

・同型の AgentSphere を使用しているグループ同士がお互いのエージェントの受け入れ先を混入させないため及び信頼していない場所からのエージェントの受け入れを制限するため。

・上記の SecurityManager によって権限に制御がかかってしまったため場合によってエージェント本来のコードを実行できないことがある。この場合は権限ではなくアクセス制限で管理することによってエージェントの自由を失わせずにセキュリティを保つためである。

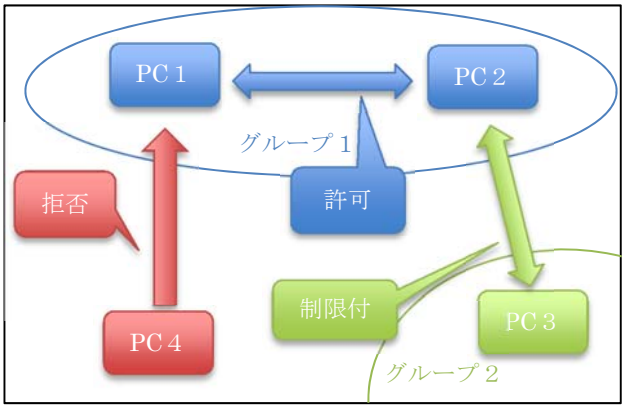

図 1　受け入れ制限の様子

受け入れ制限に実装されている方法は共通鍵暗号方式である。共通鍵を使用することでセキュリティを保つと同時に信頼する PC の識別も可能である。まずは共通かぎを用意し PC 間で共有する。共通鍵を持った状態でエージェントを PC が生成すると共通鍵で暗号化される。暗号化されたエージェントが移動すると受け入れ先の共通鍵で複合化される。この時、共通鍵が違う場合は複合化できず受け入れが拒否される。つまり、共通鍵を持った PC は既知と認証され受け入れを許可する。

上記の図では、PC1 と PC2 はお互い同じ共通鍵を有しており、なおかつお互いに移動するエージェントは SecurityManager でアクセスを制限されていない。グループ 2 も共通鍵を持っており受け入れられるが、グループ外のエージェントのため SecurityManager でアクセスが制限される。PC4 は不正アクセスをしようとするが共通鍵を有していないためアクセスが拒否される。

3. AgentSphere 用のファイルシステム

本研究では AgentSphere 用のファイルシステムの試作実装を行った。

このファイルシステムは次のような必要性があったため実装されたものである：

・エージェントがどの AgentSphere に移動しても同じファイルにアクセスする必要があるため。

・エージェントの現在位置や状況を考慮せずに使用できるファイルシステムが必要であるため。

・エージェントの権限設定によって利用できなくなったファイルアクセス機能の代替として、AgentSphere 内部で管理されたファイルアクセス機能を提供するためである。

この条件を満たすため、本研究ではファイル共有プロトコル CIFS を用いてファイルシステムの試作を行った。

3.1 ファイルシステムの使用システム

本研究でファイルシステムの実装に使用したシステムは CIFS(Common Internet File System)である。CIFS は、マイクロソフト社の開発した WindowsOS のファイル共有で使用される SMB(Server Message Block)プロトコルを拡張したものである。CIFS は TCP/IP を用いてファイル共有を行う。

この CIFS の Java ライブラリである JCIFS ライブラリを使用してファイルシステムを構築した。

3.2 ファイルシステムの実装方法

上記の JCIFS を使用すると Java をベースとする AgentSphere のエージェントも Windows の共有ファイルにアクセスすることが可能となる。しかし、SecurityManager が課す制限と競合しないように新たなクラスの実装で JCIFS の情報の媒体が作られた。JCIFS はユーザ認証情報とファイルサーバの IP アドレス、カレントディレクトリの絶対パス、ファイルの相対パスの 4 つを管理する。

これらの情報は通常 AgentSphere の本体に登録されている情報でありエージェントはこれに依存する。エージェントが別のエージェントスフィアに移動した場合はこれらの情報は維持されない。

例えばリモートファイル等をアクセスする際はカレントディレクトリの絶対パスが必要だ、パスやユーザ情報・IP アドレス等はすべてクラスに持たせることでエージェントが移動しても今まで使用していたファイルをリモートからアクセスし続けることが可能となった。

使用する中継クラスは File クラスである。このクラスの中には OutputStream クラス、InputStream クラスに相当するメソッドが入っている。これらのメソッドは内部で特権ブロックを用意しており、権限のないエージェントでも実行することができる。これらを呼び出すと、アクセスの種類の判断がまず行われる。メソッドを呼び出したエージェントの生成元 AgentSphere が、今現在の AgentSphere と同じなら、ローカルのファイルに直接アクセスする。

生成元 AgentSphere が異なる場合は File クラス内予め保存されていたユーザ認証情報、カレントディレクトリなどの情報を元にリモートのファイルアクセスを構築する。エージェントがローカルファイルにアクセスすると、その際のカレントディレクトリと IP アドレスを記憶しておく。エージェントが今後移動しても情報がクラスの中に保存されているため現在地が変わっても使用し続けることができる。

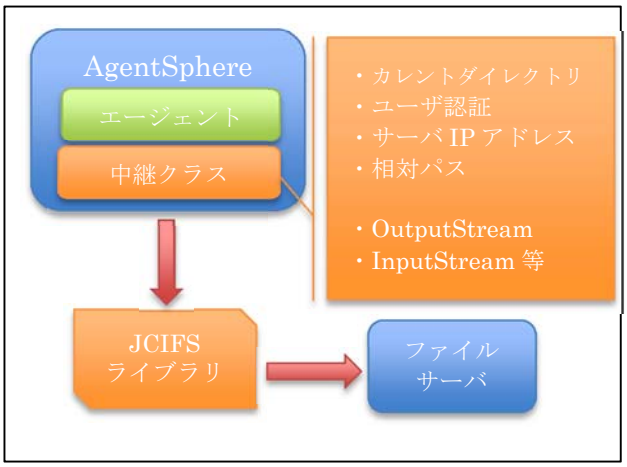

図 2　JCIFS と中継クラスの構成

4. 動作実験

4.1 権限と受け入れ制限

実装された SecurityManager を使用し外部から送られてきたエージェントの権限設定を確認するために動作実験をした。

SecurityManager を使用し、送られてきたエージェント上でローカルファイルの操作を行おうとするとエラーが出力され操作できなかった。一方で、AgentSphere 本体はファイルの入出力を実行することができた。これにより、エージェントにのみ正しくアクションの制限がかけられていることが確認できた。

エージェントの受け入れ制限が正しく機能しているかの実験も行った。AgentSphere を起動させた 3 台の PC を使用して一つのエージェントを試験的に 10 週し受け入れ制限を試した。移動先の AgentSphere が対象のエージェントを受け入れられない場合は次の AgentSphere へ移動する。そして、最後にそれぞれの AgentSphere がエージェントを受け入れた回数を表示した。

受け入れ制限をかけない場合には、各 PC に 10 回ずつ移動を行った。PC の一つに受け入れ制限をかけて他のエージェントを受け入れないように設定すると、制限されていない PC は同様に 10 回ずつ受け入れたが、制限された PC は 1 回も受け入れなかった。受け入れ制限が正しく機能していたと同時に、移動に失敗したことを認識し処理を継続できていたことが確認できた。

4.2 ファイルシステムの動作実験

ファイルシステムの動作実験には 2 台の PC、PC-A と PC-B を使用した。PC-A のローカルファイルで作業を行った後、PC-B に移動して同様にしてファイルシステムからのアクセスが可能であるか確認した。エージェント内ではローカルアクセスと同等であることを示すためにユーザ認

証情報と相対パス（test.txt）のみを指定し、カレントディレクトリと IP アドレスは自動的に取得させた。

エージェントは PC-A 上でディレクトリ内のファイルのリストの表示と test.txt への出力を行い、PC-B に移動した後、同じ test.txt の内容を読み込み表示した。PC-A 上での結果を図 3 に、PC-B 上での結果を図 4 に示す。同じアドレス上の同じファイルにアクセスできたことが確認できた。

```
15832 [Thread-11] DEBUG org.apache.commons.javaflow.bytecode.StackRecorder  - calling runnable
JCIFS Test Start.
[1] setUser
[2] file
access to: smb://133.220.114.244/Users/hasumi/Documents/AgentSphere/workspace/Primula_Eclipse/
1:.classpath
2:.project
3:.settings
4:agent
5:AgentLog.txt
6:bin
7:build
8:build.xml
9:class
10:hakuya
11:history.txt
12:lib
13:manifest.mf
14:manifest.txt
15:ModuleAgentBuild.xml
16:nbproject
17:pbuild.xml
18:PermTest
19:permtest.txt
20:ptest.txt
21:rc.dat
22:rosen.txt
23:setting
24:src
25:test.txt
26:UseJavaflowBuild.xml
[3] outputStream
2016-01-30 18:28:39,594 [DEBUG] suspend()
15887 [Thread-11] DEBUG org.apache.commons.javaflow.bytecode.StackRecorder  - suspend()
2016-01-30 18:28:39,594 [DEBUG] push reference agents.JCIFSTest@100625889/sun.misc.Launcher$App
```

図 3　PC-A の結果

```
2016-01-30 18:06:20,874 [DEBUG] suspend()
203829 [Thread-9] DEBUG org.apache.commons.javaflow.bytecode.StackRecorder  - suspend()
[4] inputStream
access to: smb://133.220.114.244/Users/hasumi/Documents/AgentSphere/workspace/Primula_Eclipse/
TestAgentから書き込みました
Success.
```

図 4　PC-B の結果

共有ファイル側の共有設定に含まれないユーザ名とパスワードではアクセスができない。別ユーザ情報を登録して上記と同様の操作を行った。

PC-A 上では図 3 と同様の結果が得られた。これは、生成元のローカルファイルであるためローカルアクセスとしてユーザ情報を用いなかったためである。

一方で、PC-B 上では図 3 のように表示された。接続先は正しいが、アクセス権限がない場合のエラーが発生した。

```
2016-01-30 18:21:55,887 [DEBUG] suspend()
23002 [Thread-9] DEBUG org.apache.commons.javaflow.bytecode.StackRecorder  - suspend()
[3] inputStream
access to: smb://133.220.114.244/Users/hasumi/Documents/AgentSphere/workspace/Primula_Eclipse/
jcifs.smb.SmbAuthException: Access is denied.
    at jcifs.smb.SmbTransport.checkStatus(SmbTransport.java:546)
    at jcifs.smb.SmbTransport.send(SmbTransport.java:683)
    at jcifs.smb.SmbSession.send(SmbSession.java:238)
    at jcifs.smb.SmbTree.send(SmbTree.java:113)
    at jcifs.smb.SmbFile.send(SmbFile.java:775)
```

図 4　PC-B の結果（情報が不足している場合）

5. 性能評価

AgentSphere は自律型並列分散処理を可能にする一般的な専門知識を有しない人および一般コンピュータ向けの環境として開発されている。しかし、長い間にわたって機能面で充実しているとはいえ一般利用についての研究があまり行われていない。

具体的には、AgentSphere を使用して既存のアルゴリズムをネットワーク上で並列分散処理に使用し、それが有意義なものであるという判断素材がない。この研究の目的は、MPI（Message Passing Interface）等の利用により C 言語で実現されている並列処理と Java で展開される AgentSphere の並列処理を比べ、“深さ優先探索”を使用した“トラベリングセールスマン問題”等の並列分散処理の恩恵を受けるアルゴリズムを比べることである。より具体的には、複数のコアを有する CPU の並列分散処理と単数の CPU を搭載した複数のコンピュータを比べて、並列分散処理を行う際の所要時間や負担を見て AgentSphere の有意性を確かめたい。

手順としては並列分散させるプログラムを自在にネットワーク上の Java 環境内に移動させて、並列処理を完了した後、結果を報告あるいは特定の端末まで運搬するという基本的な構造を実現する。この方法を使用することで限られたマシンや CPU があったとしても、より広いワールドワイドネットワーク上に展開することで性能の良いサーバやワークステーションに勝るとも劣らない性能を発揮するのかが論点となる。

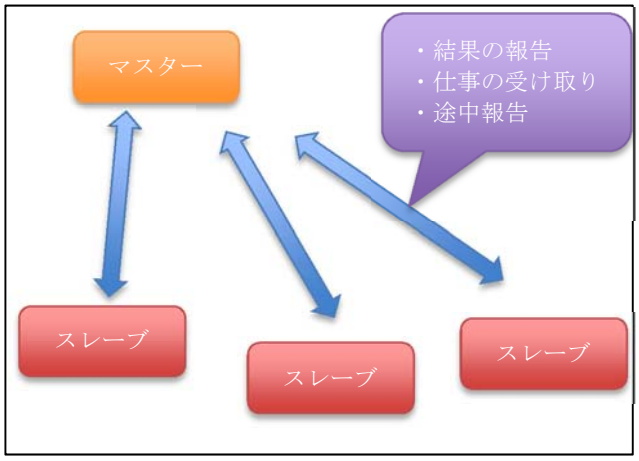

図 5　分散の様子

5.1 性能評価実験内容

実験はさまざまな実験用問題に分かれており、各問題では個別にここで使用されている AgentSphere を起動している一般コンピュータの台数および MPI のコア数は比較のため同じ数を合わせて使用している。ここではすべての問題について解説するが本命となるのが 17・18 都市の問題であるため、結果の確認にはそれらが使用されている。各問題に移る前に、一般コンピュータおよびサーバ上で何が行われるかについては下記で確認できる。

AgentSphere 用の一般コンピュータ：

性能はそれぞれ異なり、本研究では性能の差により最終結果の誤差を均一にするための手法もとっている。すべてのコンピュータには AgentSphere の環境がインストールされ、すべてが同じセグメントのローカルネットワーク上にある。使用した合計 8 台の PC のうち 1 台は必ずマスター専用端末となっている。すべてのスレーブコンピュータに移動するエージェントはマスターとのデータ通信時（最適

解の送信等）にバックアップ[2]を取るようにと設定している。

	AgentSphere 用一般 (4~8 台)	*MPI 用サーバ*
CPU	Intel Core duo, i5, i7 の混合、2~3GH z 前後	Intel(R) Xeon(R)E5-4640 @ 8core 2.4 GHz 16MB cache × 4
OS	Windows 7, 8 + JRE 8.0	Cent OS 6.5 ＋MPI-CH 3.1.2
RAM	4~16GB RAM	128GB RAM

表 1　実験環境

MPI 用サーバ

基本的な設定としては比例する一般コンピュータの数と同じプロセッサのコアを使用する。つまり AgentSphere を 4 台のコンピュータで起動させていたらサーバ上では 4 つのコアを使用してプログラムを起動することになる。なお、AgentSphere 上で実行されるエージェントについてはマルチスレッド対応ではないため各コンピュータにおいて一つ以上のコアを使用することはない。

使用したデータセットは以下の通りである：

問題	*一般コンピュータ*	*サーバ*	*TSP 都市数*
1	4 台	4 コア	8 都市
2	8 台	8 コア	8 都市
3	8 台	8 コア	14 都市
4	4 台	4 コア	17 都市
5	8 台	8 コア	18 都市

表 2　使用データセット

5.2 性能評価の結果

下記が本命となる第 4・5 問題の計測結果である。

	所要時間（4 問題目）	*所要時間（5 問題目）*
一般コンピュータサーバ	***平均 28 分***	***平均 1 時間 23 分***
	平均 33 分	平均 1 時間 30 分

表 3　問題 4・5 の結果

上記のデータからは、同じプログラムを使用して並列処理をした場合において MPI および AgentSphere の間では大した差が出ていない。これにより AgentSphere は並列処理に適していることが分かり、さらに個別の PC 同士をあたかもマルチコアの CPU と見立てた場合においてはそれに近い結果が得られることが分かる。

また、本研究では AgentSphere の耐故障性の実験も行っている。表 4 で示してある通り、項 7 の問題 4 の結果を使用して実験をしたところ、AgentSphere はバックアップ機能を利用することで途中クラッシュしてしまった場合において最初から処理を開始せずに途中の結果を引き継ぐことができる。それに比べて MPI は単一サーバなので最初から処理を開始するしかない。耐故障性と可用性について AgentSphere はバックアップができるのでより有用である。

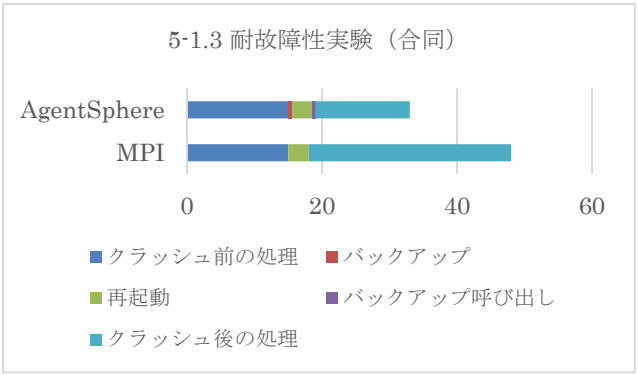

表 4　耐故障性の結果（単位：分）

6. おわり

本研究によって AgentSphere に基本的なセキュリティ対策である SecurityManager および CIFS を用いたファイルシステムが実装された。これにより不正なエージェントの受け入れの制限が可能となり信頼している相手とのみ更新をすることができた。

また、ファイルシステムの実装によりエージェントがどこに移動してもファイルにアクセスすることが可能となった。さらにセキュリティ権限の実装でアクセスしたとしても権限を持たないファイルへのアクセスが制限された。

AgentSphere は同アルゴリズムを使用した複数コアのサーバと比べても処理の性能に大した差が見受けられず複数の PC をネットワークおよび AgentSphere 同士でつなぐことであたかもマルチコアの環境として扱うことが可能であるため有用であることが分かった。

参考文献

[1] JCIFS. https://jcifs.samba.org/
[2] 加藤 史彬,田久保 雅俊,櫻井 康樹,甲斐 宗徳「コード変換による強マイグレーション化モバイルエージェントの実現」,FIT2007, B-024, Sep.2007