Vol.2015-ICS-179 No.6
2015/3/20

知的分散エージェントのための スナップショット管理機構の構築

岸上 友樹[1,a)]　打矢 隆弘[1]　内匠 逸[1]　木下 哲男[2]

概要：分散環境において動的にサービスの構成を適応するシステムの設計手法として，エージェント指向コンピューティングが注目されている．エージェント指向コンピューティングでは，エージェントシステムと呼ばれる知識情報に応じて自律的に動作するソフトウェアを用いたシステムによってユーザにサービスを提供する．現在，エージェントシステムの開発・運用を行うために様々なフレームワークが存在する．本研究では，数あるフレームワークの中で DASH と呼ばれるエージェントフレームワークについて取り扱う．従来の DASH には分散環境で動作するエージェントシステムの再利用を目的とした機構が存在しておらず，エージェントシステムの継続利用が困難となる場合があった．そこで，利用中・利用後のエージェントをデータベースに保存して管理し，保存時と同じ状態でエージェントシステムを再利用するための機構を構築する．この機構の実装により，DASH における AS の継続利用をフレームワーク側から支援し，エージェントの永続性向上を図る．

1. はじめに

近年，インターネット等の広域分散処理の発展に伴い，ユーザのソフトウェアやサービスに求める要求や利用形態が多様化している．そこで，サービス要求に柔軟に対応できるシステムの設計手法として，エージェントシステム (以降 AS) が注目されている．エージェントとは，人間の様々な仕事を代行するソフトウェアの総称である．エージェントはユーザの操作を介さずに他のエージェント，データベース，WEB システムなど，様々なコンポーネントと協調して動作することで，ユーザの代わりに問題の解決を図り，サービスを提供する．

本研究では，エージェントの動作基盤となるフレームワークの一つである DASH(Distributed Agent System based on Hybrid architecture)[1][2] について取り扱う．DASH は AS の運用環境として用いられており，AS の効率的な利用を支援している．DASH は目的を達成するために動的に AS を組織するため，状況に応じて柔軟にシステムの形態を変更することができる．環境に順応した AS の構成や，それぞれのエージェントが持つ情報は，AS の継続的な利用に不可欠なものであるが，従来の DASH ではその保存・管理が十分に支援されていない．そこで，利用中・利用後のエージェントをデータベースに保存・管理し，エージェントの継続利用を行うための機構を提案する．この機構の実装により，AS の永続性向上を実現し，DASH における AS 運用の利便性を向上させる．

[1] 名古屋工業大学大学院 工学研究科 情報工学専攻，名古屋市
Nagoya Institute of Technology, Gokiso-chou, Syowa-ku, Nagoya-shi, 466-8555 JAPAN
[2] 東北大学電気通信研究所，仙台市
Research Institute of Electrical Communication, Tohoku University, 2-1-1 Katahira, Aoba-ku, Sendai-shi, 980-8577 JAPAN
[a)] kishigami@uchiya.nitech.ac.jp

2. エージェントフレームワーク DASH

2.1 DASH の基本的な構成

DASH はリポジトリ型エージェントフレームワークと呼ばれ，作成されたエージェントを保存しておくサーバであるリポジトリと，エージェントが実際に動作する環境であるワークプレースによって主に構成されている．DASH におけるユーザへのサービス提供を図 1 に示す．

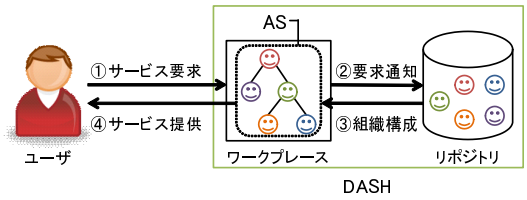

図 1　DASH におけるサービス提供の流れ
Fig. 1　Operation flow of DASH agents.

ユーザのサービス要求がワークプレースを介してリポジトリに伝えられ，サービス要求に応じてリポジトリ内の

エージェントを組織構成し，ワークプレース上で動作させることでサービスが提供される．

AS の動作を行うにあたり，エージェントが互いの名前と存在するワークプレースを把握しなければならない場合がある．そのような場合には，ネームサーバと呼ばれるデータストア用サーバが用いられる．ネームサーバを用いることで，ネットワークで接続されたリポジトリとワークプレースに存在するエージェントの名前や機能を把握することができる．

2.2 DASH エージェントの動作方法

各エージェントは元々与えられていた情報や，他のエージェントからのメッセージ (DASH メッセージ) 等によって得た知識情報 (ファクト) を，エージェント毎に持つ格納場所 (ワーキングメモリ) に保持する．また，if-then 型のルール (図 2) 集合を動作知識として有し，これらのルールによってエージェントの動作を決定する．ルールの条件部に該当したファクトが存在する場合，そのルールのアクション部を実行する．エージェントがマッチングを繰り返して何らかの動作を行っている状態を推論状態と呼ぶ．また，実行できるルールがなくなった場合は，他のエージェント等からメッセージを受け取るなど，新たにファクトを得るまでは動作ができないため，メッセージの受け取り待ちをする．この状態をメッセージ待ち状態と呼ぶ．

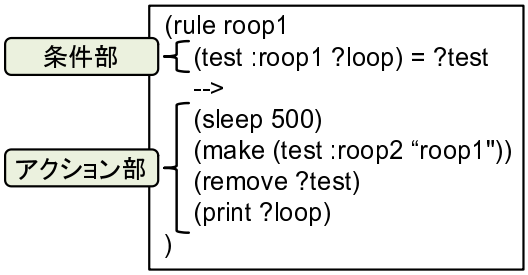

図 2 DASH エージェントのルール記述例
Fig. 2 Rule description of DASH agent.

また，if-then 型のルールのアクションに用意されていない動作を行う場合は，ベースプロセスと呼ばれる Java プロセスを用いて自由度の高い動作を可能とする．

2.3 DASH の問題点

従来の DASH には以下のような問題点が挙げられる．

(P1) 利用後の AS の継続利用が困難

AS は利用後に消去されるため，以前利用した AS の情報を継続して利用することができない．また，再び同じ AS を動作させるためには，AS を組織しなおす必要があり，動的に組織される AS の以前の構成と同じ状態にするためには時間を要する．このように，従来の DASH には AS の継続的な利用に支障がある．

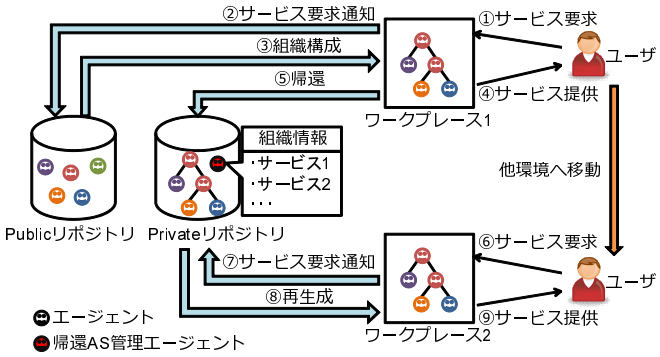

図 3 帰還型エージェントフレームワークの概要図
Fig. 3 Design of circulation agent framework.

3. 関連研究

3.1 帰還型エージェントフレームワーク

伊藤らの関連研究 [3] では，DASH における永続性についての支援の向上を目的として「ユビキタス環境用帰還型エージェントフレームワーク」が開発された．この研究では DASH を基盤として，動的に組織され環境に適応した AS をリポジトリに移動 (帰還) して保持し，エージェントの継続利用を支援する機構を導入している (図 3)．この機構の実装により，以前利用した AS を異なる環境で再使用する事が可能となっている．

3.2 関連研究の問題点

関連研究によって，DASH におけるエージェントの継続利用のフレームワークによる支援が行われた．しかし，関連研究においても AS を実運用する上で，以下のような問題が挙げられる．

(P2) 分散環境で動作する AS の帰還は不可能

AS は一つのワークプレースだけでなく，複数のワークプレース上に跨ってエージェントを配置することで組織を構成可能である．しかし，関連研究で実装された AS の帰還方法では，ユーザが利用しているワークプレースにあるエージェントのみを対象としている．よって，分散環境で動作する AS に対しては帰還を行うことができない．

(P3) 利用中の AS の不意の消滅に対処不可能

計算機の電源断等によって，予定外に消滅してしまった AS を再度同じ状態で利用するための機構を，関連研究の機構は有していない．リアルタイムに動作を行っている AS にとっては，持っている知識情報の消失に繋がり，継続利用を行うことが困難となる．

4. 提案機構の設計

前章で示した問題点を“エージェントスナップショット管理機構”を導入することで解決を図る．本章では，提案機構の設計について説明する．

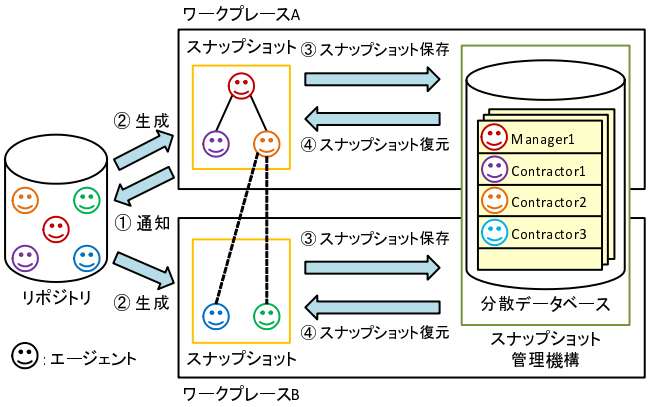

図 4　提案機構の概要図

Fig. 4　Operation flow of proposed mechanism.

4.1　提案機構の機能

提案機構の有する機能を以下に示す．

(F1) データベースによるエージェント情報の管理機能

提案機構の概要図を図 4 に示す．サービス提供を終了した AS のエージェント情報をデータベースに保存して管理することで，再利用したいエージェントの消失を防ぐ．この際，エージェント情報としてデータベースに格納する情報群をスナップショットと呼ぶ．

エージェントを再利用する際は，データベースに保存されたスナップショットを基に，エージェントを保存時の状態で生成することでエージェントの復元を行う．AS のサービス提供終了のタイミングは自動的に判断することができないため，サービス終了時にエージェントを削除する前にユーザが保存指示を GUI 上で行う．

この機能により，AS のサービスを再利用する際は，データベースから復元することで保存時の状態から継続して利用することが可能となる．AS を始めから組織し直す方法に比べて，時間的に効率化できるだけでなく，AS の実行途中で行われるエージェントの増減によって動的に組織された AS の構成と知識情報をそのまま継続利用できる．これらの実装によって，従来の DASH において十分でなかったエージェントの永続性の向上を実現する．

(F2) スナップショット自動取得によるバックアップ機能

AS の動作中に一定時間毎にエージェントのスナップショットを取得してデータベースに格納することで，AS が不意に消滅した場合の継続利用を可能とする．復元を行う際は，直近数回分の保存日時から復元ポイントを選択することにより，保存時と同様の状態で AS を継続して利用することができる．

4.2　分散環境における制約

AS は，異なる計算機を含めた複数のワークプレース上にエージェントを配置して動作する場合が存在する．そのため，エージェントの保存を行う際は，それぞれの環境に存在するエージェントを全て保存しなければならない．しかし，エージェントに対して操作を行うことができるのは，エージェントが存在するワークプレースからのみである．そのため，複数のワークプレースが協調して AS の保存を行う必要がある．

5.　提案機構の実装

提案機構の実装方法として，スナップショットの取得方法，エージェント保存時の同期について説明する．

5.1　スナップショットの取得

5.1.1　スナップショットの内容

スナップショットとして保存を行う内容は，各エージェントが持つエージェント名，ルール群，ファクト，AS 内の親エージェント子エージェントの関係性，ルールの発火履歴，エージェント間メッセージ，ベースプロセス等が挙げられる．これらの情報を用いることで，エージェントを保存時と同じ状態で復元することが可能となる．

5.1.2　スナップショットの文字列化

本研究でスナップショットの保存に用いるデータベースは，文字列のみを保存可能なデータベースである Cassandra[4] を用いる．そのため，上記のスナップショットをバイト列にシリアライズして取得する必要がある．DASH で用いられるエージェントやベースプロセス，DASH メッセージ等は Java で記述されている．よって，スナップショットは Java プログラムのままシリアライズを行うことで，バイト列に変換可能である．ただし，シリアライズできない情報 (transient，static が指定されたフィールドなど) は取得できないため，スナップショットからエージェントを復元した際に，改めて値を与える必要がある．

5.1.3　スナップショット取得のタイミング

推論状態のエージェントからスナップショットを取得するためには，エージェントが実行するルールの合間である必要がある．ルールのアクション部を実行している途中でスナップショットを取得した場合，復元後にアクション部の内容を読み飛ばしてしまう可能性がある (図 5)．エー

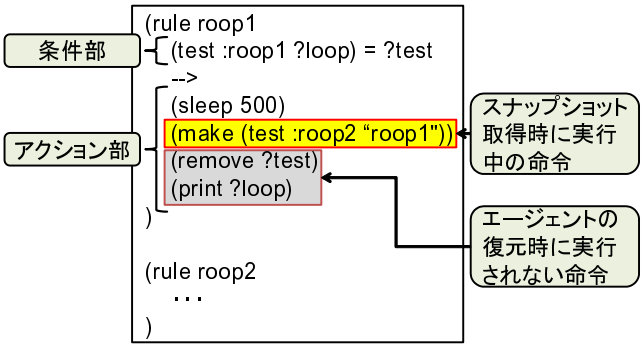

図 5　推論状態のエージェントのスナップショット取得

Fig. 5　Acquiring snapshot of agent on interference state.

ジェントの動作履歴はルール単位であり，スナップショット取得時にアクション部の一部の命令が実行されていなくても，ルール自体は実行されたという履歴が残ってしまうためこの問題が発生する．

提案機構では，エージェントからのスナップショット取得を確実にルールの合間で行うため，スナップショット取得直前にエージェントの動作停止を行う．具体的な保存手順として，エージェントがフレームワークから保存命令を受けてからデータベースに保存されるまでに行われる 3 つの工程を説明する．

1. 保存対象エージェントを停止

スナップショット取得命令を受けたエージェントは，ルールを実行し終えた後に次のルール実行に移る前に動作を停止する．

2. スナップショットを取得

エージェントの情報をスナップショットとして取得する．取得する際には，復元に必要な情報をシリアライズして，データベースに格納できるバイト列に変更する．

3. 保存対象エージェントを再始動

停止していたエージェントの動作を再開する．エージェントが停止してから再始動するまでの時間は，エージェントがスナップショット取得操作によって動作を阻害されている時間となる．

5.2 分散環境におけるエージェント保存時の同期

複数のエージェントを同時に保存する際には，エージェント間で保存情報の同期を取る必要がある．例として，一方のエージェントがメッセージ送信後にスナップショットを取得し，メッセージを受け取るエージェントがメッセージ受信前にスナップショットを取得すると，エージェントの復元時にメッセージが受信できないといった場合が考えられる．

そこで提案機構では，エージェント間の通信である DASH メッセージを対象として，スナップショット取得時にエージェント間で同期を取るためのアルゴリズムを導入する．

5.2.1 Chandy・Lamport のアルゴリズム

Chandy・Lamport のアルゴリズム [5] は，多くの分散スナップショットアルゴリズムを構築する上で参考にされている基礎的なアルゴリズムである，

前提条件

Chandy・Lamport のアルゴリズムを利用するための前提条件として以下の条件が必要となる．

- 各ノード間の通信が FIFO である．
- 隣接するノード (1 ホップで通信可能なノード) との双方向通信が可能である．
- 保存対象がアルゴリズム実行中に増減しない．

初期状態

各ノードの状態はスナップショット未取得と取得済の 2 通りの状態を取り，初期状態はスナップショット未取得である．また，各ノードはメッセージリストを保持し，ノードの復元の際に改めて受信するメッセージを管理する．

アルゴリズムの流れ

(1) 任意のノードひとつでスナップショット取得を行い，ノードの状態をスナップショット取得済に変更する．
(2) マーカをすべての隣接ノードに送信する．
(3) マーカを受信したノードはスナップショットを取得し，ノードの状態をスナップショット取得済に変更する．
(4) 隣接ノードに対してマーカメッセージを送信する．
(5) 全てのノードにおいて隣接ノードとマーカを交換したらアルゴリズムを終了する．

上記アルゴリズムを行っている間，スナップショットの取得前後にノードがマーカ以外のメッセージを受信する場合がある．その際は，メッセージの送信ノードと受信ノードがスナップショット取得済か否かで 4 通りに場合分けして処理することで，スナップショット取得時の同期を取る．Chandy・Lamport のアルゴリズムにおけるメッセージ処理を，時空ダイアグラム (図 6) を用いて説明する．

図に用いられているアルファベットの意味を以下に示す．

- c：各ノードのスナップショットを取得したタイミングを曲線でつないだもの (マーカの流れ)
- Ac，Bc，Cc：各スナップショット取得時刻
- Pc：ノードの状態が保存される前の時間帯
- Fc：ノードの状態が保存された後の時間帯

1. Pc から Pc へのメッセージ (メッセージ A)

メッセージの送信と受信が，それぞれのノードのスナップショットを取得する前に行われている．よって，スナップショットを取得した際に，それぞれのノードではメッセージの送受信の内容を反映した状態となっており，メッセージを保存する必要はない．

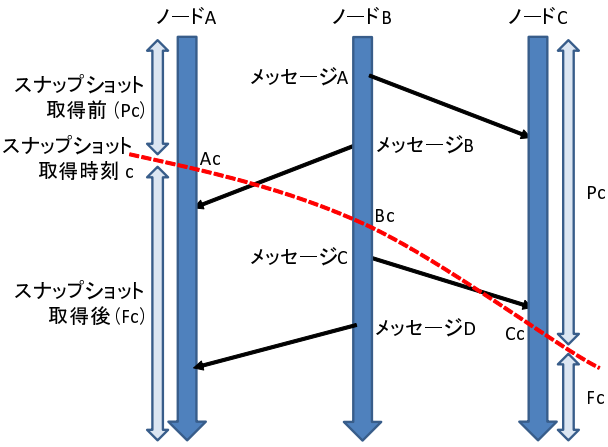

図 6 Chandy・Lamport のアルゴリズムの時空ダイアグラム
Fig. 6 Spacetime diagram of the Chandy-Lamport algorithm.

2. Pc から Fc へのメッセージ (メッセージ B)
メッセージを送信したという情報は，スナップショット取得時刻 Bc において，ノード B におけるスナップショットで取得できている．しかし，メッセージを受信したという情報は，スナップショット取得時刻 Ac において，ノード A におけるスナップショットで取得できていない．そのため，ノード A におけるスナップショットを取得してから受信したメッセージをリストで格納し，復元時にそのメッセージを受信することで，スナップショット取得前後のメッセージの送受信を矛盾無く行う．

3. Fc から Pc へのメッセージ (メッセージ C)
スナップショット取得時刻 Bc において，メッセージを送信したという情報がノード B に無いにもかかわらず，スナップショット取得時刻 Cc においてメッセージを受信したという情報がノード C に存在する．この場合，復元後に改めてメッセージが送信されてしまい，二重にメッセージを受信することとなる．よって，この状態が発生した場合はスナップショットに矛盾があるといえる．しかし，通信が FIFO であるという前提の場合，各ノード間のマーカの後に送信されたメッセージが，マーカを追い越して送信相手が受信することは無い，そのため，Chandy・Lamport のアルゴリズムにおいてはこのような状況は発生し得ない．

4. Fc から Fc へのメッセージ (メッセージ D)
メッセージの送信とメッセージの受信が，それぞれのスナップショットを取得した後に行われている．よって，ノードを復元した後でメッセージの送信と受信を改めて行うため，メッセージの保存を行う必要はない．

5.2.2 提案機構への適応

Chandy・Lamport のアルゴリズムを提案機構に導入するため，DASH におけるノードを定義する．

提案機構において，ノードは保存対象のエージェントとなる．しかし，エージェントは基本的に全エージェントと通信できるため，保存対象となるエージェントの増加に従い，隣接ノードとして扱うノードが爆発的に増加してしまう．そこで，エージェント間でのメッセージ送受信を行う際に，メッセージが必ずワークプレースを介している点に着目して，この問題を解決する．

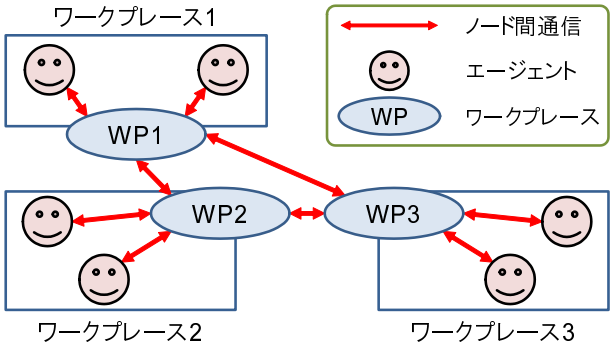

図 7 DASH におけるノード群
Fig. 7 Nodes on DASH.

DASH におけるノード群の概要図を図 7 に示す．保存対象となるエージェントをノードとして定義し，それらの隣接ノードを各エージェントが存在するワークプレースのみとする．そして，ワークプレース同士も隣接ノードとして定義する．このようにノード群を設定することでマーカ数を抑制し，効率良くスナップショットを取得する．

6. 提案機構の性能評価

提案機構の性能評価のため，エージェントフレームワーク JADE[6][7] に存在するエージェント保存機能である persistence[8] との機能充実度の比較と保存速度比較を行った．また，従来のスナップショット管理機構 [9] との保存速度の比較を行い，エージェント保存時の動作阻害時間の確認を行った．

6.1 JADE との機能・性能比較

実験概要

JADE の持つエージェント保存機能である persistence は，提案機構と同様にデータベースにエージェントを保存して管理し，必要に応じて復元することができる機能である．比較はフレームワークによって支援されているかを確認しており，エージェント開発者がエージェントに機能を追加することにより実装する場合を考慮しない．比較項目を以下に示す．

- 保存対象環境
  単一環境・分散環境のどちらでも保存を行えるか．ユーザの操作する計算機とは異なる計算機上で動作するエージェントの保存を行って確認する．
- 自動保存
  ユーザの命令を介さない自動的な保存が行えるか．自動保存機能を有しているかを確認する．
- エージェント間同期
  通信し合うエージェント間で同期を行っているか．メッセージを交換し合うエージェントの保存・復元によって確認する．
- 動作阻害時間
  エージェント保存時に動作を阻害する時間はどの程度か．単一環境上での 10 体のエージェント保存時の平均動作阻害時間を求める．保存を行うエージェントは，DASH と JADE 共に何も行動を行わないルールを繰り返し実行するエージェントを利用した．

結果・考察

実験の結果を表 1 に示す.

表 1 DASH と JADE の機能・性能比較
Table 1 Preformance Comparison between DASH and JADE.

	DASH(提案機構)	JADE
保存対象環境	ネットワーク全体	自計算機上のみ
自動保存	可能	不可能
エージェント間同期	有り	無し
動作阻害時間	4.28 ミリ秒	1.95 ミリ秒

保存対象環境においては，JADE では計算機を跨った AS の保存をフレームワークで支援していないことが確認された．よって，複数の環境に跨って動作する AS を消去後にも継続的に利用を行う場合，エージェントの開発者が分散環境でも保存できるように実装する必要があり，DASH に比べてユーザの手間がかかることとなる．この点では，DASH は JADE と比較して優位であるといえる．

自動保存においては，JADE ではエージェントの自動保存機能をフレームワークで支援していないことが確認された．よって，計算機の再起動等によるエージェントの不意の消滅に対して対処するためには，エージェントの開発者が外部ファイルに情報の保存を行う等の記述が必要となる．この点では，DASH は JADE と比較して優位であるといえる．

エージェント間同期においては，JADE では複数エージェント保存時のエージェント間同期をフレームワークで支援していないことが確認された．よって，エージェントが協調して動作を行う場合には，復元後にエージェント間で情報の食い違いが生まれる可能性がある．この点では，DASH は JADE と比較して優位であるといえる．

動作阻害時間においては，JADE は DASH でのエージェント保存と比較してエージェントの動作阻害時間が半分程度であることが確認された．この違いは，保存を行う対象であるエージェントのデータサイズの差異によるものである．同じ機能を有するエージェントに対して保存を行った場合でも，DASH と JADE でそれぞれ専用のエージェントを用いており，DASH の方がデータサイズが大きかったため，スナップショット取得に時間がかかる結果となった．この点では，DASH は JADE と比較して優位でないといえる．

これらの比較実験の結果から，DASH は JADE と比較して多くの機能がフレームワークによって支援されていることが確認できた．一方，保存時に発生する動作阻害時間は JADE が勝ることが確認された．AS は複数の環境上で同時に動作することが多くあり，それらを復元時に情報の食い違いなく保存できることは非常に有用であると考えられる．よって動作阻害時間の短さが重要であるような AS を除いて，DASH によるエージェント保存の方が有用であるといえる．

6.2 従来のアルゴリズムとの保存速度比較

実験概要

分散環境で動作する AS の保存に適したアルゴリズムの導入により，エージェント動作阻害時間が短縮されたことを確認するため，従来手法と保存速度の比較を行った．従来手法は，単一環境で動作を行う AS を対象としたプロトタイプであり，分散環境で動作する AS の保存には適していなかった．保存対象となるエージェントは，自計算機上のエージェント 5 体と他計算機上のエージェント 5 体との計 10 体である．各エージェントの停止開始から再始動までを計測し，その内訳をエージェント 1 体毎の平均で示す.

結果・考察

それぞれの計測結果を表 2 に示す.

表 2 エージェント保存時間の内訳 (ms)
Table 2 Breakdown of the time of Saving Agent.(ms)

操作	従来手法	新手法
非動作阻害時間		
エージェント停止操作	1.54	0.68
動作阻害時間		
スナップショット取得操作	5.78	4.16
エージェント再始動操作	0.04	0.05
通信/操作間インターバル	38.80	0.07
小計	44.62	4.28
合計	46.16	4.96

- エージェント停止・スナップショット取得操作
  従来手法では，保存対象エージェント間で同期を取るために，各エージェントでエージェント停止・スナップショット取得操作が同じタイミングで実行されるように設計されていた．このため，エージェント停止・スナップショット取得操作に最も時間を要したエージェントに合わせて，多くのエージェントに待機時間が発生していた．一方，新手法では個別にエージェント停止・スナップショット取得操作を行っていくため，他のエージェントの動作に関わらず次の操作が行える．この違いにより，操作に要する時間の平均値で新手法が勝る結果となった.
- エージェント再始動操作
  各エージェントの再始動に要する時間がほとんど変わらないため，手法による違いは見られなかった.
- エージェント停止中の通信/操作間インターバル
  従来手法ではエージェント停止後にエージェント停止・スナップショット取得・エージェントの再始動のタイミングを同期させるための通信をワークプレー

ス間で行っており，非常に長い時間を要していた．一方，新手法では各エージェントで操作を個別に行っている．エージェント間同期は復元後のメッセージ受信で実現しているため，停止中に行われる通信時間分が動作阻害時間が短縮された．

分散スナップショットアルゴリズムを用いたことにより，分散環境で動作する AS に対するエージェントの保存時に，動作阻害時間が短縮されたことを確認した．特に時間を要した通信時間が削減されたおり，分散環境での提案機構の実用性が向上した．

7. おわりに

本研究では，エージェントフレームワーク DASH における永続性に関する問題点を挙げ，その解決手法としてエージェントの情報をデータベースに保存して管理するスナップショット管理機構を開発した．

提案機構と JADE の persistence との比較を行い，同時保存数・保存対象環境・自動保存の面で提案機構の有用性を示した．一方，保存の際に生じるエージェントの動作を阻害する時間においては，persistence が勝る結果となった．また，提案機構の初期プロトタイプとエージェント保存時の動作阻害時間をした結果，大幅に短縮されていることが確認できた．

提案機構を実装したことにより，様々なエージェントに対してデータベースによるスナップショットの管理が行えるようになり，エージェントの永続性が向上したといえる．そして，エージェントシステムの継続的な AS の利用を支援し，エージェントシステムの利用拡大を促進するものと思われる．

参考文献

[1] 打矢 隆弘，武田 敦志，菅沼 拓夫，木下 哲男，“エージェントフレームワークにおけるリポジトリ機構の設計と実装”，情報処理学会論文誌，Vol.44，No.3，pp.799-811，2003.
[2] DASH ユーザマニュアル，http://www.ka.riec.tohoku.ac.jp/idea/html/index.html
[3] 伊藤 翔太，打矢 隆弘，内匠 逸，“ユビキタス環境用帰還型エージェントフレームワークの設計”，情報科学技術フォーラム講演論文集 ，Vol.9，No.2，pp.369-372，2010.
[4] Eben Hewitt 著，大谷 晋平，小林 隆 訳．“Cassandra”．O’REILLY JAPAN，2011.
[5] Chandy K. Mani, and Leslie Lamport, “Distributed Snapshots: Determining Global States of Distributed Systems”, ACM Transactions on Computer Systems, Vol.3, Issue 1, pp.63-75, 1985.
[6] Fabio Bellifemine, Giovanni Caire and Dominic Greenwood, “Developing Multi-Agent Systems with JADE”, john Wiley & Sons,Ltd, 2007.
[7] Jade - Java Agent DEvelopment Framework, http://jade.tilab.com/
[8] JADE Object Management and Persistence in Java, http://www.jade.co.nz/downloads/jade/papers/jade_wp_javapersistence.pdf
[9] 岸上 友樹，打矢 隆弘，内匠 逸，木下哲男，“エージェントフレームワーク DASH におけるスナップショット管理機構の構築”，第 75 回全国大会講演論文集，pp.375-376，2013.