情報処理学会論文誌 Vol.59 No.2 812–820 (Feb. 2018)

分散協調とローカル対応を融合した緊急タスク割当

林 久志[1,†1,a)]

受付日 2017年6月26日, 採録日 2017年11月7日

概要：近年，多くのシステムがネットワークで接続され，システムオブシステムズを構成している．大規模災害では多くの故障要因が広範囲で発生し，一定時間内に修理しないと分散配置された多くのシステムが故障に至る．限られた数のリソースを使って広範囲で修理をして故障要因を除去するためには，修理を行うエージェントどうしが協調してタスク割当を決めることが重要である．しかしながら，協調には時間を要するため，限られた時間内で故障要因を除去できずに故障に至る場合もある．一方，見つけた故障要因をその場で修理（ローカル対応）すれば即応性は向上するが，広範囲の故障要因を万遍なく修理することは難しくなる．本論文では，故障まで残された時間に応じて協調とローカル対応を使い分け，人の確認を省略する残時間考慮ハイブリッド型アルゴリズムを提案する．協調手法としては，耐故障性に優れる分散協調型のタスク割当アルゴリズムをベースにする．

キーワード：マルチエージェントシステム，タスク割当，分散協調とローカル対応，緊急対応，耐故障性

Emergency Task Allocations that Reconcile Decentralized Coordination and Local Response

Hisashi Hayashi[1,†1,a)]

Received: June 26, 2017, Accepted: November 7, 2017

Abstract: In recent years, many systems are connected through the network, which constitute systems of systems. If large disasters repeatedly happen, multiple causes of failures are created. In order to remove causes of failures using limited resources, it is vital to coordinate agents that are in charge of repairing. Especially, decentralized coordination has tolerance against single point failures. However, it takes time to coordinate agents. In this paper, we present a new task allocation algorithm for emergency situations. This algorithm combines decentralized coordination and local response, which significantly reduces system failures.

Keywords: multi-agent system, task allocation, coordination and local response, emergency response, fault tolereance

1. はじめに

ネットワークで接続され，分散配置されたマルチエージェントシステム（MAS）において，災害時の耐故障性を高めることは重要な課題である．特に大規模災害においては，災害イベントが繰り返し発生し，多くの場所でシステムの故障要因が発生する．効率的に修理をして故障要因を除去しないと，多くのエージェントが故障し，MAS 全体が機能しなくなってしまうため，限られた数のリソースを効果的に活用し，限られた時間で効果的に修理タスクを割り当てることが重要となる．

同時多発的に発生する故障要因を効果的に修理するためには，各エージェントが協調して修理を分担し，取り組む

[1] 株式会社東芝研究開発センターシステム技術ラボラトリー
System Engineering Laboratory, Corporate R&D Center, Toshiba Corporation, Kawasaki, Kanagawa 212–8582, Japan
[†1] 現在，公立大学法人首都大学東京産業技術大学院大学産業技術研究科
Presently with School of Industrial Technology, Advanced Institute of Industrial Technology, Tokyo Metropolitan University
[a)] hayashi-hisashi@aiit.ac.jp

本研究は株式会社東芝で実施された．

必要がある．しかしながら，協調してタスク割当を行うためには時間を要する．故障要因発見から故障に至るまでの時間が短い場合には，タスク割当のための時間を減らすことが必要である．タスク割当のための時間を減らすためには，エージェント間で協調せずに反射的にローカルで修理（ローカル対応）を行ったり，人による確認時間を省略したりすることが効果的である．本論文では，故障まで残された時間に応じて協調とローカル対応を使い分け，人による確認をするか否かを決定する残時間考慮ハイブリッド型のタスク割当アルゴリズムを提案し，他のアルゴリズムと比較する．

文献 [1], [2], [3] で議論されているように，タスク割当は大まかに集中協調型と分散協調型に分類される．集中協調型では，1 つの管理エージェントのみが配下の各エージェントから情報を収集し，タスクとエージェントの組合せを考え，各エージェントにタスクを割り当てる．一方，分散協調型では，複数の管理エージェントが互いに協調してタスク割当を決定する．多くの既存のアルゴリズムは集中協調型であるが，分散協調型は多少の数の管理エージェントが故障してもシステム全体としては機能し続けるため，頑強である．本研究では，エージェントの耐故障性を高める必要があるので，分散協調型に着目する．

契約ネットプロトコル [4] のようなオークションアルゴリズムは計算時間が少なく，動的タスク割当 [5], [6], [7], [8], [9], [10] によく用いられる．また，オークションアルゴリズムは分散協調型として実装することも容易である．そのため，本研究では契約ネットプロトコルをベースにした分散協調型のタスク割当アルゴリズムを拡張・修正していくこととする．遅延時間が発生する場合にほぼ同時に多くのオークションが行われると単純な契約ネットプロトコルではうまく機能しないため，本論文ではその点も改良する．

本論文は以下のように構成される．2 章で関連研究について議論する．3 章で想定する MAS と問題設定を説明する．4 章で 3 つのアルゴリズムを定義する．5 章でシミュレーション設定を詳細に説明する．6 章でシミュレーション結果を示し，分析する．7 章で本論文をまとめる．

2. 関連研究

タスクとエージェントの組合せを求めるために，メタヒューリスティックスを用いた最適化計算をすることがある．文献 [11], [12] では，タスク割当のためのいくつかのメタヒューリスティックスが比較されている．文献 [12] では，タブー探索が他のアルゴリズム（GA や ACO など）よりも計算時間や最適性の面で良い解を出すことがシミュレーション実験で示されている．文献 [11] では，計算時間が限られているときには，PSO が他のアルゴリズム（GA や貪欲法など）よりも最適性の面でやや良い解を出すことがシミュレーション実験で示されている．しかしながら，これらのメタヒューリスティックスを用いたアプローチは一般的に計算時間がかかることが知られている．

文献 [2], [13] では，分散制約最適化（DCOP）の分野で有名な Max-Sum アルゴリズムを用いて，タスク割当問題を解いている．エージェント間の接続情報を表すグラフを用いる他の DCOP のアルゴリズムと比較すると，Max-Sum はエージェントの故障に頑強である．しかしながら，DCOP のアルゴリズムでは，エージェント間で多数のメッセージが繰り返し送信され，通信と計算のための時間がかかる．

文献 [3] では，将来のエージェントの故障確率を考慮したタスク割当を行っているが，故障しても修理はしない．文献 [14] では，故障に備えて，バックアップエージェントを用意している．しかしながら，特にハードウェアをともなうエージェントのバックアップ作成コストは高い．文献 [15] では，故障に頑強なエージェントのチームを作成する．基本的なアイデアは，複数の能力を持つ必要以上の数のエージェントをあらかじめチームにいれておくことにより，エージェントの故障時には，そのエージェントのタスクを他のエージェントがカバーする．

3. 想定する MAS と問題設定

想定する MAS は複数の**単位 MAS** からなるものとする．単位 MAS の各機能は異なるハードウェア上で実現されることもあり，それぞれの機能は単独で故障することもあるので，これらの機能は独立したエージェントとする．

単位 MAS は，**図 1** のとおり，故障要因情報を検知する 1 個の**センシングエージェント**と，故障要因を除去する修理アクションを実行する 0 個以上の**アクション実行エージェント**と，故障要因情報をもとにアクション実行エージェントに修理タスクを割り当てる 1 個の**管理エージェント**からなる．各単位 MAS の管理エージェントどうしがネットワークで接続されている場合には，各管理エージェントは他の単位 MAS の管理エージェントに修理タスクを割り当てることもできる．故障要因を修理アクションで除去せずに一定時間が経過すると，ランダムに選択される 1 個のエージェントが一定確率で故障する．

センシングエージェントは故障発生時刻までの残時間が一定時間になると，一定確率でその故障要因情報をセンシ

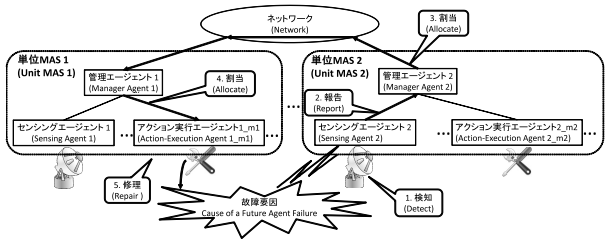

図 1 MAS アーキクチャ
Fig. 1 MAS architecture.

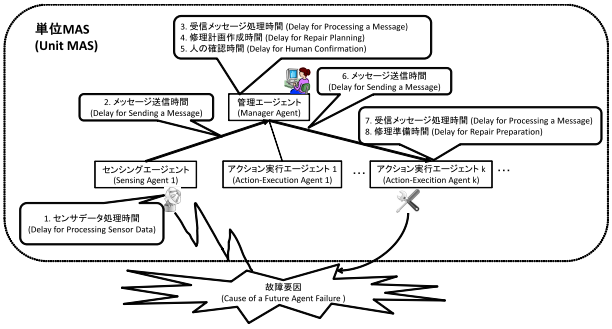

図 2 遅延時間
Fig. 2 Delay times.

ングして管理エージェントに報告する．アクション実行エージェントは初期状態でリソースが割り振られており，リソース数が 1 以上のときに修理アクションを実行できるが，1 つのアクション実行部は同時に 2 つ以上の修理アクションを実行できない．管理エージェントはこれらの制約を考慮のうえ，アクション実行エージェントに故障要因の修理タスクを割り当てる．

管理エージェントから修理タスクを割り振られたアクション実行エージェントは，故障発生時刻までの残時間が一定時間以内になると管理エージェントからの指示により修理アクションを開始し，一定確率で故障要因を除去する．修理アクションを実行した場合，実行結果（故障要因除去の成功・失敗）にかかわらず，リソースを 1 つ消費する．

本論文では，時間が重要となる問題を扱うため，各種遅延時間を考慮する．すなわち，**図 2** のように，センシングエージェントがセンサデータを処理して故障要因を検知する時間，他のエージェントにメッセージを送信する時間，他のエージェントからメッセージを受信して処理する時間，管理エージェントが修理計画を作成する時間，管理エージェントのオペレータ（人）が確認する時間，アクション実行エージェントが修理の準備をする時間を考慮する．

4. アルゴリズム

本章では，修理タスクを割当・実行する 4 つのアルゴリズムを定義する．1 つめのローカル対応型アルゴリズムでは，各単位 MAS の管理エージェント間で情報交換を行わず，故障要因を発見した単位 MAS で修理する．2 つめの分散協調型アルゴリズムでは，新しい故障要因情報を入手した管理エージェントが，契約ネットプロトコル [4]（の改良版）に基づき，他の単位 MAS の管理エージェントと情報交換したうえで 1 つの単位 MAS の管理エージェントを選択し，修理タスクを割り当てる．3 つめの残時間考慮ハイブリッド 1 型アルゴリズムと 4 つめの残時間考慮ハイブリッド 2 型アルゴリズムは，時間があるときには，2 つめの分散協調型アルゴリズムを用いるが，時間がないときには，1 つめのローカル対応型アルゴリズムを用いる．それでも時間がない場合には，残時間考慮ハイブリッド 2 型アルゴリズムでは，オペレータ（人）の確認時間も省略する．

2 つめと 3 つめと 4 つめのアルゴリズムでは，修理に失敗した場合には，そのタスクの再割当を行う．また，遅延時間の間に，ほぼ同時に同じリソースに複数のタスクが割り当てられた場合にも，割り当てられなかったタスクの再割当を行う．また，複数の単位 MAS から同じ単位 MAS にタスク割当が過度に集中することが原因で再割当が頻繁に起きないないように，可能ならば，一定時間は同一の単位 MAS に対してタスクを割り当てないように工夫している．一方，1 つの単位 MAS 内では，1 つの管理エージェントがアクション実行エージェントの予約管理をしているため，同じアクション実行エージェントに同時に複数の修理アクションを割り当てることはない．

なお，これらのアルゴリズムにおいて，オペレータ（人）の確認や修理計画作成のステップでは，本論文ではそれらの具体的方法までは定めていない．本アルゴリズムでは，これらのステップは遅延時間を生じるという意味のみを持つ．

以下に，単位 MAS どうしで協調しないローカル対応型アルゴリズムを定義する．

Algorithm 1（ローカル対応型） 各エージェントは故障していなければ以下のように動作する：

- センシングエージェント
  (1) 新しい故障要因を検知したときには，その故障要因情報を同じ単位 MAS に所属する管理エージェントに（故障していなければ）報告する．
- 管理エージェント
  (1) 同じ単位 MAS に所属するセンシングエージェントから新しい故障要因情報を入手したときには，同じ単位 MAS に所属するアクション実行エージェントのうち，故障しておらず，かつ，リソース数が 0 でなく，かつ，修理アクションの予約がされていないものがあれば，1 つ選択し，以下を行う．
    (a) 選択したアクション実行エージェントに対し，その故障要因の修理予約を行う（修理予約の管理は管理エージェントが行うものとする）．
    (b) 修理計画を作成する．
    (c) オペレータ（人）が確認する．
    (d) 修理予約のある各故障要因に対し，故障発生時刻の一定時間前[*1]になったら，修理予約したアクション実行エージェントにその故障要因の修理アクションの実行を命令し，その修理予約の情報を消去する．ただし，そのアクション実行エージェントが故障しているときには何もしない．

[*1] 遅延時間を考慮して修理開始可能時刻以降のできるだけ早い時間に修理を開始できるようにアクション実行命令を出す．

- アクション実行エージェント
  (1) 同じ単位 MAS に所属する管理エージェントから命令を受けたら，指定された故障要因に対する修理アクションを実行し，その実行結果によらず，リソースを 1 消費する．
  (2) 修理アクションの実行結果（成功または失敗）を管理エージェントに（故障していなければ）報告する．

以下に，単位 MAS どうしで情報交換して意思決定する分散協調型アルゴリズムを定義する．

Algorithm 2（分散協調型） センシングエージェントとアクション実行エージェントは故障していなければ Algorithm 1 と同様に動作する．管理エージェントは故障していなければ以下のように動作する：

- 管理エージェント
  (1) 同じ単位 MAS に所属するセンシングエージェントから新しい故障要因情報を入手したときには，以下のように動作する．
    (a) 各単位 MAS の故障していない管理エージェントに対し，その故障要因の修理タスク受入可否と修理成功確率を聞く．
    (b) もし，その修理を実行できる単位 MAS の管理エージェントが存在すれば，以下のように動作する：
      (i) もし，過去一定時間[*2]の間に他の修理を依頼していない単位 MAS が存在するならば，それらの単位 MAS の中から，そうでない場合には，その他の単位 MAS の中から，最も修理成功確率[*3]の高い単位 MAS の管理エージェントを 1 つ選択する．
      (ii) 選択した管理エージェントにその修理タスクを割り当てる．
  (2) 自他の単位 MAS の管理エージェントから修理タスクを割り当てられたときには，以下のように動作する：
    (a) 同じ単位 MAS に所属するアクション実行エージェントのうち，故障しておらず，かつ，リソース数が 0 でなく，かつ，修理アクションの予約がされていないものがあれば，1 つ選択し，以下のように行動する．
      (i) Algorithm 1 の管理エージェントのステップ 1a からステップ 1d と同様にしてアクション実行エージェントを 1 つ選択し，修理アクションの実行を命令する．
      (ii) 修理タスクの実行結果を通知され，その結果が失敗であったときには，ステップ 1a から 1b と同様にして，依頼された修理タスクの再割当を行う．
    (b) そうでなければ，ステップ 1a から 1b と同様にして，依頼された修理タスクの再割当を行う．

以下に，残時間が十分にあれば分散協調し，残時間が十分にない場合にはローカル対応する残時間考慮ハイブリッド 1 型アルゴリズムと，さらに時間がなければ人の確認も省略する残時間考慮ハイブリッド 2 型アルゴリズムを定義する．

Algorithm 3（残時間考慮ハイブリッド 1 型） センシングエージェントとアクション実行エージェントは故障していなければ Algorithm 1 と同様に動作する．管理エージェントは故障していなければ以下のように動作する：

- 管理エージェント
  (1) 同じ単位 MAS に所属するセンシングエージェントから新しい故障要因情報を入手したときには，もしくは，修理タスクの再割当を行うときには，以下のように動作する．
    (a) 各遅延時間を考慮しても故障発生までに修理を完了することが可能ならば，Algorithm 2 の管理エージェントのステップ 1a からステップ 1b と同様にしてその修理タスクを自他の管理エージェントに割り当てる．
    (b) そうでない場合には，自分自身にその修理タスクを割り当てる．
  (2) 自他の単位 MAS の管理エージェントから修理タスクを割り当てられたときには，Algorithm 2 の管理エージェントのステップ 2 と同様にしてアクション実行エージェントを 1 つ選択し，修理アクションの実行を命令する．

Algorithm 4（残時間考慮ハイブリッド 2 型） Algorithm 3 と同様に動作する．ただし，管理エージェントのステップ 2 において，人が確認すると修理を故障発生までに完了することが不可能ならば，オペレータ（人）による確認は省略する．

5. シミュレーション設定

本章では前章のアルゴリズムを評価するためのシミュレーション実験で用いる詳細設定を説明する．

本評価シナリオは，文献 [6] に記載のアプリケーション（Weapon-Target Assignment）に関連している．文献 [6] では，Target が故障要因に相当し，Target を検知するレーダはセンシングエージェントに相当し，Weapon が故障要因を除去するアクション実行エージェントに相当する．設定値は，我々が想定しているアプリケーションにおける典型的な値を用いる．

*2 本論文の実験では，1 分間に設定する．
*3 以前の関連研究 [9], [16] では，最も早く修理開始できる単位 MAS を選択していたが，後の我々の研究で，最も修理成功確率の高い単位 MAS を選択したほうが，故障数が減ることが判明している．

5.1 単位 MAS の配置パターン

表 1 にシミュレーション実験で配置する単位 MAS，エージェント，リソースの数を示す．この配置では，表中の UMAS 1 から UMAS 5 までの 5 種類の単位 MAS を用いる．故障要因を早くセンシングしたり，早く修理できたりする高性能な単位 MAS は高コストなので少なめに配置し，低性能だが低コストである単位 MAS は多めに配置する．単位 MAS を多めに配置すれば，より多くの故障要因に同時に対応できるようになる．このようなバランスを考慮し，これらの単位 MAS を，それぞれ，1，2，2，4，8 ずつの数だけ配置する．なお，各単位 MAS の故障要因の検知性能と修理性能は次の節で示す．合計の単位 MAS の数は 17（$=1+2+2+4+8$）である．各単位 MAS には必ず管理エージェントとセンシングエージェントが 1 個ずつ存在するとする．管理エージェントやセンシングエージェントの合計数も 17 である．

UMAS 1 から UMAS 6 までの各単位 MAS のアクション実行エージェントの数は，それぞれ，0，0，4，2，1 とする．つまり，合計のアクションエージェントの数は 24（$=2\times4+4\times2+8\times1$）である．初期状態において，UMAS 3，UMAS 4，UMAS 5 の各アクション実行エージェントが持つリソース数は，それぞれ，6，4，8 であり，総リソース数は 144（$=2\times4\times6+4\times2\times4+8\times1\times8$）である．UMAS 1 と UMAS 2 はアクション実行エージェントを持たない．そのため，これらの単位 MAS のセンシングエージェントが探知した故障要因の情報は UMAS 3，UMAS 4，UMAS 5 でしか活用できない．単独のアクション実行エージェントは同時に複数の修理アクションを実行することはできないが，各アクション実行エージェントは同時に修理アクションを実行できることとする．

ローカル対応型アルゴリズムでは，UMAS 1 や UMAS 2 の高性能なセンシングエージェントが検知した故障情報を活用することはできないが，他の単位 MAS がこれらの単位 MAS の故障要因を除去することは可能である．他のアルゴリズムでは，単位 MAS 間の通信により，UMAS 1 や UMAS 2 で検知した故障情報を活用することができる．これは単位 MAS 間通信の大きなメリットであり，本シミュレーション設定により，このメリットも評価結果に反映される．

表 1 単位 MAS，エージェント，初期リソースの数
Table 1 # of unit MASs, agents, and initial resources.

単位 MAS の種別	配置される単位 MAS の数	各単位 MAS の構成			
		管理エージェントの数	センシングエージェントの数	アクション実行エージェントの数	初期リソース数
UMAS 1	1	1	1	0	-
UMAS 2	2	1	1	0	-
UMAS 3	2	1	1	4	24(6)
UMAS 4	4	1	1	2	8(4)
UMAS 5	8	1	1	1	8(8)
計	17	17	17	24	144

( ) 各アクション実行エージェントの初期リソース数

5.2 故障要因の検知性能と修理性能

表 2 に各単位 MAS のセンシングエージェントが各故障要因の検知を故障の何秒前から開始するのかまとめる．同じ故障要因に対しては UMAS 1 が最も早く検知し，UMAS 5 が最も遅く検知する．すなわち，UMAS 1, ..., UMAS 5 はセンシング性能の高いものから低い順に並べている．なお，故障要因 1 はセンサの性能によらず故障の直前（一定時間前）まで検知できない．故障要因 2 は検知してから故障に至るまでの時間が短い．故障要因 3 は検知してから故障に至るまでの時間がさらに短い．

表 3 に各単位 MAS のアクション実行エージェントが各故障要因を除去するための修理を故障の何秒前から開始可能なのかまとめる．同じ故障要因に対しては UMAS 3 が最も早くから修理を開始でき，最も遅く修理を開始できるのは UMAS 5 である．すなわち，UMAS 3，UMAS 4，UMAS 5 は，修理開始性能の高いものから順に並べている．

表 4 に各単位 MAS のアクション実行エージェントが故障発生予定時刻の x 秒前から修理を開始したときに，修理に何秒かかるのかまとめる．単位 MAS の種別によって修理時間に差はないが，故障要因の種別によって修理時間は異なる．我々が想定するアプリケーションでは，一定速度で故障要因が近づいているときに，修理開始と同時に

表 2 故障要因検知開始時間（故障の何秒前から故障要因を検知可能か）
Table 2 Time to start detecting a failure cause.

単位 MAS の種別	故障要因の種別		
	Cause 1	Cause 2	Cause 3
UMAS 1	43.2 s	120.0 s	42.4 s
UMAS 2	43.2 s	60.0 s	21.2 s
UMAS 3	43.2 s	24.0 s	8.5 s
UMAS 4	43.2 s	14.4 s	5.1 s
UMAS 5	18 s	6 s	2.1 s

表 3 修理開始可能となる時間（故障の何秒前から修理開始可能か）
Table 3 Time to start repairing a failure cause.

単位 MAS の種別	故障要因の種別		
	Cause 1	Cause 2	Cause 3
UMAS 3	36.0 s	12.0 s	4.2 s
UMAS 4	18.0 s	6.0 s	2.1 s
UMAS 5	10.8 s	3.6 s	1.3 s

表 4 故障 x 秒前に修理を開始したときの修理所要時間
Table 4 Repair time when starting the repair x seconds before an agent failure.

単位 MAS の種別	故障要因の種別		
	Cause 1	Cause 2	Cause 3
UMAS 3	x/2.5 s	x/4.5 s	x/9.5 s
UMAS 4	x/2.5 s	x/4.5 s	x/9.5 s
UMAS 5	x/2.5 s	x/4.5 s	x/9.5 s

表 5 故障要因検知確率と修理成功確率
Table 5 Probability of successful detection and repair.

故障要因検知確率	修理成功確率
90%	平均値 80%，標準偏差 10%の正規分布（100%以上は 100%，0%以下は 0%と見なす）

表 6 遅延時間
Table 6 Delay times.

センサデータ処理時間	1 s
メッセージ送信時間	1 s
受信メッセージ処理時間	1 s
修理計画時間	5 s
人の確認時間	0, 10, 15, 20 s
修理準備時間	5 s

一定速度で修理手段を故障要因の場所に送る．Cause 1，Cause 2，Cause 3 は，故障に至る速度の遅いものから順に並べている．修理は早くから開始するほど長く時間がかかるが，それでも早く始めた方が早く終わる．修理手段が故障要因に到達した時点で修理は終了する．

表 5 に故障要因検知確率（故障要因の発生後に，各単位 MAS のセンシングエージェントがその故障要因を探知できる確率）と修理成功確率（各単位 MAS のアクション実行エージェントが故障要因の修理をしたときに修理に成功し，その故障要因が除去される確率）をまとめる．故障要因検知確率はすべて 90%である．修理成功確率は正規分布に従って変化し，平均値は 80%で標準偏差は 10%である．ただし，100%以上の場合は 100%と見なし，0%以下の場合は 0%と見なす．

5.3 遅延時間

本論文のシミュレーションシナリオでは，遅延時間によって修理が間に合わないことがあるため，**表 6** に示される各種遅延時間を考慮する．センシングエージェントが故障要因を認識する際に 1 秒のセンサデータ処理時間を要する．次に，エージェント間でメッセージを送信するのに 1 秒かかり，メッセージを受信したエージェントはそのメッセージを処理するのに 1 秒かかる．管理エージェントはアクション実行エージェントに修理アクションの実行を命令する前に修理計画を作成する必要があるが，それに 5 秒かかる．管理エージェントのオペレータ（人）が修理計画を確認するが，人の確認時間は 0 秒，10 秒，15 秒，20 秒と変化させて評価する．アクション実行エージェントは修理アクションの実行を開始するまでに，5 秒の準備時間を要する．

これらの値は，我々が想定するアプリケーションの現状システムでは，もっと遅い場合もあったり，ネットワーク接続すらされていない箇所もあったりするが，将来システムを想定した仮定値を用いる．

表 7 災害イベント発生パターン
Table 7 Occurrence patterns of disasters events.

災害イベントの発生回数	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
災害イベントの発生間隔	1 h
災害イベント時に発生する故障要因数	10
故障要因発生間隔	1 s
修理しない場合の故障発生確率	90%
故障要因の各種別の発生確率と，故障要因発生から故障に至るまでの時間	Cause 1：60%，1,800 s Cause 2：30%，600 s Cause 3：10%，212 s

5.4 災害イベントと故障要因の発生パターン

表 7 に，今回のシミュレーションで想定する災害イベントと故障要因の発生パターンを示す．災害イベントが発生すると合計 10 の故障要因が 1 秒間隔で発生する．Cause 1，2，3 の故障要因は，それぞれ，60%，30%，10%の確率で発生し，故障要因が発生してから故障に至るまでの時間は，それぞれ，1,800 秒，600 秒，212 秒である．災害イベントは十分長い時間間隔（1 時間）を空けて，10 回発生させる．すなわち，故障要因は合計 100 発生[*4]する．修理アクションで故障要因を除去しない場合，90%の確率でランダムに選択されたエージェントが故障する．

6. シミュレーション結果

本章ではシミュレーション結果を示し，考察を行う．本実験では，前章で説明したシミュレーション設定を用いる．シミュレーションでは，各事象（故障要因発生，故障要因検知，修理成功・失敗，故障要因が除去されない場合の故障発生）の起き方を変化させる．具体的には，各事象を表 5 と表 7 に示す各確率に基づき乱数を発生させる．このとき，乱数のシードを変えることにより，各事象の起き方を変化させる．本実験では，3 種類のアルゴリズムを人の確認時間（0 秒と 10 秒と 15 秒と 20 秒の 4 種類）を変えて，各パターンで 1,000 回ずつ合計 12,000（$= 3 \times 4 \times 1{,}000$）回のシミュレーションを試行する．各シミュレーションの

[*4] 故障要因数 100 よりも初期リソース数 144 のほうが多いが，アクション実行エージェントや管理エージェントが故障すると配下のリソースは利用できなくなるので，初期リソース数は十分とはいえない．

試行では，異なる乱数のシードを用いる．なお，修理の成功・失敗については正規分布を，その他は一様分布の乱数となり，その発生には Java の Random クラスを用いる．本研究の目的は修理により故障要因を除去し，故障に至るエージェントの数を減らすことにあるので，評価軸としては以下を用いる：

- 平均故障数（= 全シミュレーションの試行におけるエージェントの故障数の合計/シミュレーションの試行回数）
- 平均修理成功数（= 全シミュレーションの試行における修理成功数の合計/シミュレーションの試行回数）

図 3 と**図 4** と**図 5** と **図 6** に人の確認時間を，それぞれ，0 秒，10 秒，15 秒，20 秒と変化させたときの平均故障数を示す．また，**図 7** と**図 8** と**図 9** と **図 10** に人の確認時間を，それぞれ，0 秒，10 秒，15 秒，20 秒と変化させたときの平均修理成功数を示す．

人の確認時間を増やすことは，タスク割当のために使える時間が減ることを意味する．x 軸では災害イベント発生回数を 1 から 10 まで変化させる．つまり，発生する故障要因数は 10 から 100 まで変化させる．y 軸ではエージェントの平均故障数を表す．

全般的にローカル対応型アルゴリズムの平均故障数が最も多く，平均修理成功数が最も少ないことが分かる．残時間考慮ハイブリッド 2 型アルゴリズムの平均故障数が最も

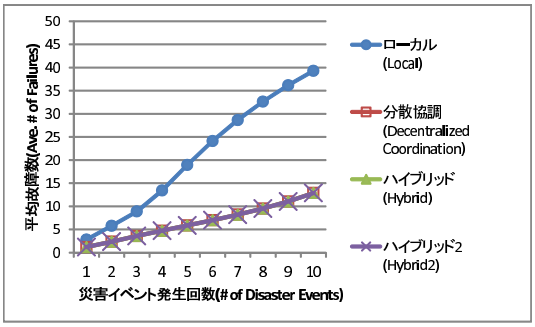

図 3 平均故障数（人の確認時間 0 秒の場合）
Fig. 3 Average number of failures (Human confirmation time is 0 s).

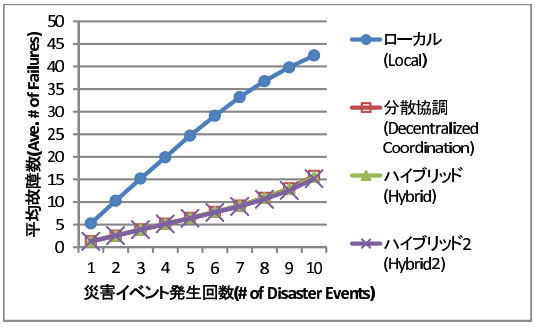

図 4 平均故障数（人の確認時間 10 秒の場合）
Fig. 4 Average number of failures (Human confirmation time is 10 s).

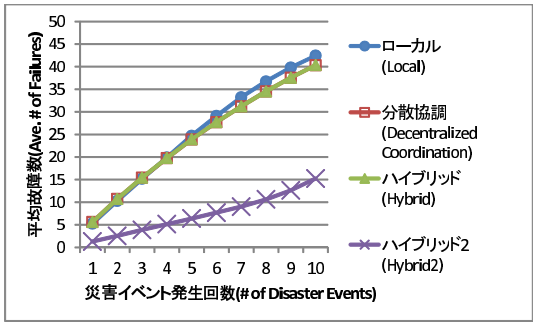

図 5 平均故障数（人の確認時間 15 秒の場合）
Fig. 5 Average number of failures (Human confirmation time is 15 s).

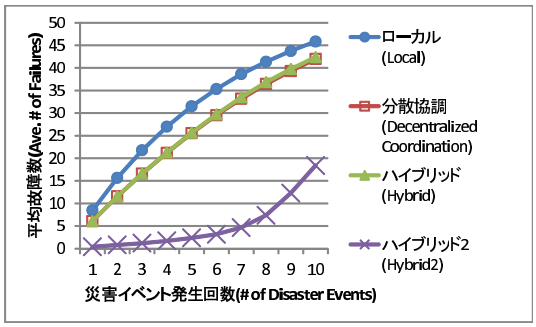

図 6 平均故障数（人の確認時間 20 秒の場合）
Fig. 6 Average number of failures (Human confirmation time is 20 s).

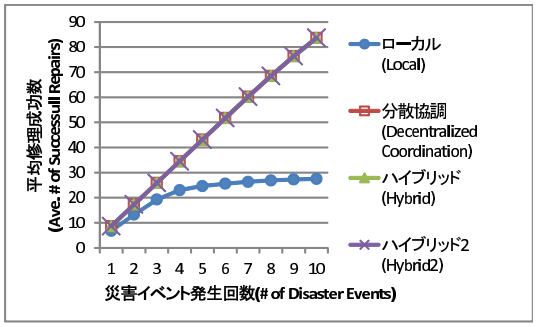

図 7 平均修理成功数（人の確認時間 0 秒の場合）
Fig. 7 Average number of successful repairs (Human confirmation time is 0 s).

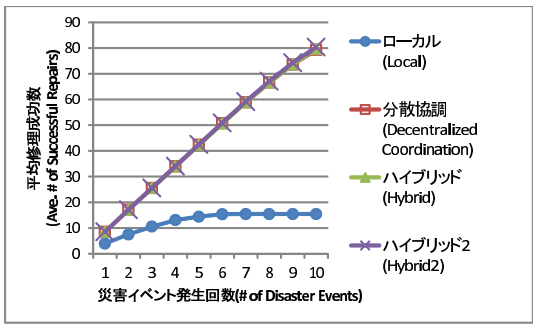

図 8 平均修理成功数（人の確認時間 10 秒の場合）
Fig. 8 Average number of successful repairs (Human confirmation time is 10 s).

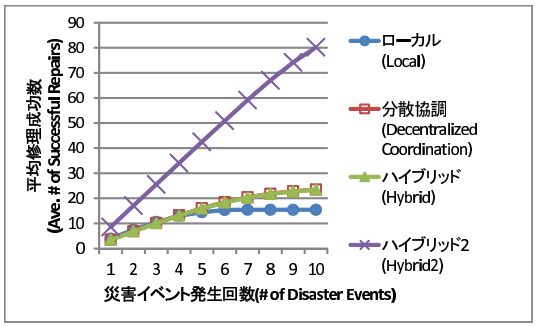

図 9 平均修理成功数（人の確認時間 15 秒の場合）

Fig. 9 Average number of successful repairs (Human confirmation time is 15 s).

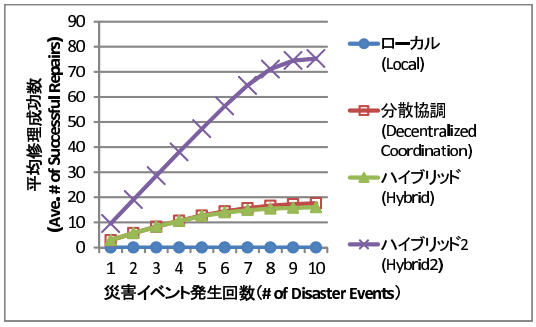

図 10 平均修理成功数（人の確認時間 20 秒の場合）

Fig. 10 Average number of successful repairs (Human confirmation time is 20 s).

少なく，平均修理成功数が最も多いことが分かる．どの場合でもローカル対応型アルゴリズムの平均故障数が多く，平均修理成功数が少ないのは，単位 MAS どうしで協調して修理しないため，同じ故障要因に対し異なる複数の単位 MAS で同時に修理をして無駄にリソースを消費し，早い段階でリソース切れになることと，まったくアクション実行エージェントが割り当てられなかった故障要因による故障が多く発生することが原因である．

図 3 と図 4 と図 7 と図 8 に着目すると，ローカル対応型アルゴリズムと比較して，分散協調型アルゴリズムと残時間考慮ハイブリッド 1 型アルゴリズムと残時間考慮ハイブリッド 2 型アルゴリズムの平均故障数が大幅に減り，平均修理成功数が大幅に増え，分散協調の効果が高いことが分かる．分散協調型アルゴリズムと残時間考慮ハイブリッド 1 型アルゴリズムと残時間考慮ハイブリッド 2 型アルゴリズムの平均故障数や平均修理成功数の差が少ないのは，分散協調のための時間が十分にとれる場合が多いため，残時間考慮ハイブリッド型アルゴリズムでローカル対応する機会が少なくなるからである．

図 5 と図 6 と図 9 と図 10 に着目すると，分散協調型アルゴリズムや残時間考慮ハイブリッド 1 型アルゴリズムと比較して，残時間考慮ハイブリッド 2 型アルゴリズムの平均故障数が大幅に減り，平均修理成功数が大幅に上がることが分かる．これは，ローカル対応しなければ修理に間に合わない場合が多く，また，この場合において，人による確認時間も省略してはじめて修理に間に合う場合が多いことを意味する．

以上をまとめると，本シミュレーション実験では，どの場合でも残時間考慮ハイブリッド 2 型アルゴリズムが最も効果的であることが分かる．

7. まとめ

本論文では，4 種類の修理タスク割当アルゴリズム（ローカル対応型，分散協調型，残時間考慮ハイブリッド 1 型，残時間考慮ハイブリッド 2 型）をマルチエージェントシミュレーションにより評価した．分散協調型と残時間考慮ハイブリッド 1 型と残時間考慮ハイブリッド 2 型のアルゴリズムにおいては，遅延時間があっても同時に多くの災害イベントに対応できるように改良した．特に残時間考慮ハイブリッド 2 型アルゴリズムに関しては，故障までの残時間が短い状況において，単位 MAS 間の協調や人の確認を省略して反射的にローカル対応することにより，大幅にエージェントの故障数を減らすことに成功した．

本論文では，限られた数のシナリオを用いてシミュレーション評価を実施したが，本論文のアルゴリズムは多くの応用分野で有用であると考えられる．今後は，より多くのシナリオを用いて評価を行いながら本アルゴリズムを改良していくとともに，様々な応用分野への適用を目指していく．

参考文献

[1] Choi, H.-L., Brunet, L. and How, J.: Consensus-Based Decentralized Auctions for Robust Task Allocation, *IEEE Trans. Robotics*, Vol.25, No.4, pp.912–926 (2009).

[2] Macarthur, K., Stranders, R., Ramchurn, S. and Jennings, N.: A Distributed Anytime Algorithm for Dynamic Task Allocation in Multi-Agent Systems, *Proc. AAAI Conf. Artificial Intelligence*, pp.701–706 (2011).

[3] Rahimzadeh, F., Khanli, L. and Mahan, F.: High Reliable and Efficient Task Allocation in Networked Multi-Agent Systems, *Autonomous Agent and Multi-agent Systems*, Vol.29, No.6, pp.1023–1040 (2015).

[4] Smith, R.: The Contract Net Protocol: High-level Communication and control in a Distributed Problem Solver, *IEEE Trans. Computers*, Vol.C-29, No.12, pp.1104–1113 (1980).

[5] Ahmed, A., Patel, A., Brown, T., Ham, M., Jang, M.-W. and Agha, G.: Task Assignment for a Physical Agent Team via a Dynamic Forward/Reverse Auction Mechanism, *Proc. Int. Conf. Integration of Knowledge Intensive Multi-Agent Systems*, pp.311–317 (2005).

[6] Beaumont, P. and Chaib-draa, B.: Multiagent Coordination Techniques for Complex Environments: The Case of a Fleet of Combat Ships, *IEEE Trans. Systems, Man and Cybernetics - Part C*, Vol.37, No.3, pp.373–385 (2007).

[7] Chen, J., Yang, J. and Ye, G.: Auction Algorithm Approaches for Dynamic Weapon Target Assignment Prob-

lem, *Proc. Int. Conf. Computer Science and Network Technology*, pp.402–405 (2015).
[8] Gerkey, B. and Matarić, M.: Sold!: Auction Methods for Multirobot Coordination, *IEEE Trans. Robotics and Automation*, Vol.18, No.5, pp.758–768 (2002).
[9] Hayashi, H.: Comparing Repair-Task-Allocation Strategies in MAS, *Proc. Int. Conf. Agents and Artificial Intelligence*, Vol.1, pp.17–27 (2017).
[10] Lagoudakis, M., Markakis, E., Kempe, D., Keskinocak, P., Kleywegt, A., Koenig, S., Tovey, C., Meyerson, A. and Jain, S.: Auction-Based Multi-Robot Routing, *Proc. Int. Conf. Robotics: Science and Systems*, pp.343–350 (2005).
[11] Johansson, F. and Falkman, G.: Real-Time Allocation of Firing Units to Hostile Targets, *Jour. of Advances in Information Fusion*, Vol.6, No.2, pp.187–199 (2011).
[12] Xin, B., Chen, J., Zhang, J., Dou, L. and Peng, Z.: Efficient Decision Making for Dynamic Weapon-Target Assignment by Virtual Permutation and Tabu Search Heuristics, *IEEE Trans. Systems, Man, Cybernetics - Part C*, Vol.40, No.6, pp.649–662 (2010).
[13] Ramchurn, S., Farinelli, A., Macarthur, K. and Jennings, N.: Decentralized Coordination in RoboCup Rescue, *The Computer Jour.*, Vol.53, No.9, pp.1447–1461 (2010).
[14] Guessoum, Z., Briot, J., Faci, N. and Marin, O.: Toward Reliable Multi-Agent Systems: An Adaptive Replication Mechanism, *Multiagent and Grid Systems*, Vol.6, No.1, pp.1–24 (2010).
[15] Okimoto, T., Schwind, N., Clement, M., Riberio, T., Inoue, K. and Marquis, P.: How to Form a Task-Oriented Robust Team, *Proc. Int. Conf. Autonomous Agents and Multiagent Systems*, pp.395–403 (2015).
[16] Hayashi, H.: Integrating Decentralized Coordination and Reactivity in MAS for Repair-Task Allocations, *Proc. Int. Conf. Practical Applications of Agents and Multi-Agent Systems*, pp.95–106 (2017).

林 久志 （正会員）

1995 年早稲田大学理工学部情報学科卒業．1996 年インペリアルカレッジロンドン大学院計算学研究科修士課程修了．1999 年同大学院電気電子工学研究科博士課程修了．Ph.D. 1998～1999 年同大学院同研究科研究助手．2000～2017 年 9 月株式会社東芝研究開発センターでエージェントと人工知能の研究に従事．2017 年 10 月より公立大学法人首都大学東京産業技術大学院大学産業技術研究科創造技術専攻准教授．人工知能学会，電子情報通信学会，日本ソフトウェア科学会各会員．