URL　https://journal.ntt.co.jp/article/41223
DOI　https://doi.org/10.60249/26115002

Actor Modelを用いた分散AI推論効率化技術

車両や道路インフラに搭載されたカメラ・センサの観測情報をAI（人工知能）で組み合わせる交通監視では，観測対象や車両の移動に伴い，必要なAIや連携相手が変化します．また，分散環境でAI処理を実行する場合，適切な実行場所も変化します．こうした変化に追随して効率良くAI処理を行うために，AI処理を，状態を持つActorとして表現し，動的なActor間連携と配置，状態を維持した再配置を実現する分散AI推論効率化技術に取り組んでいます．本稿では，Actor Modelを用いるねらいと技術の特徴，交通監視への適用例を解説します．
キーワード：#分散AI，#Actor Model，#動的リソース配置

大久保　淑実　/坂本　啓
星野　玲那　/横田　謙祐
福島　卓弥　/外山　将司
NTTコンピューティングテクノロジーセンタ

実世界の協調認識と分散AI処理

近年，車両や道路インフラへのカメラ・センサの搭載が進み，複数の観測情報をAI（人工知能）で組み合わせて活用する環境が広がっています．例えば，交差点では固定カメラだけでなく，周辺を走行する車両のカメラも利用することで，単一のセンサではとらえにくい死角を補い，歩行者や自転車などをより確実に認識できます．このように，複数のセンサやAIを協調させて対象を認識すること（協調認識）で，交通安全をはじめとする実世界の認識サービスの高度化が期待されます．

こうしたサービスでは，カメラやセンサがさまざまな場所に分散して存在します．映像などの大容量データをすべて1カ所へ集約すると，通信量や通信遅延が大きくなる場合があります．そのため，センサに近いエッジサーバ*1でAI処理を行い，必要に応じてクラウドの計算資源も利用するなど，複数の計算環境でAI処理を実行する分散構成が有効です．

さらに，観測対象やセンサの移動，計算環境の負荷変動により，解析に利用するAIの組合せと適切な実行場所は時間とともに変化します．このため，実世界の状況変化に追随しながら，必要なAIを連携させ，適切な計算環境で継続的に実行する仕組みが必要です．

協調認識を支えるActor Model

■固定的な処理パイプラインの課題

複数のAIを組み合わせる方法の1つとして，入力，推論，結果統合などの処理順序をあらかじめ定めたパイプラインがあります．構成要素や関係が固定されている場合には分かりやすい方法ですが，実世界を対象とした協調認識では，解析に必要な要素とその関係自体が変化します．

このような変化をパイプラインの組替えとして扱うと，構成要素が増えるほど管理が複雑になります．そこで私たちは，実世界の構成要素や解析処理を，それぞれ独立した処理主体として表現する方法に着目しています．新しい対象が加われば対応する処理主体を生成し，必要な相手と連携させ，対象がなくなれば終了します．実世界における主体の増減と関係の変化を，処理主体の増減と連携関係の変化として表現する考え方です．

■Actor Modelによる分散AI処理の表現

このような処理モデルと親和性が高いのが，分散システムのプログラミングモデルの1つであるActor Modelです．Actor Modelでは，システムを独立したActorの集合として表現します．各Actorは自身の内部状態を持ち，他のActorから受け取ったメッセージを契機に処理し，状態を更新したり別のActorへメッセージを送ったりします．

AI解析では，センサに対応する処理，観測対象を追跡する処理，解析

＊1 エッジサーバ：利用者や機器の近くに設置し，データを低い遅延で処理するためのサーバです．

結果を集約する処理などをActorの振る舞いとして表現できます．各Actorは状態を持ちながら継続的に動作し，異なる計算環境へ配置しても同じモデルで連携できます．

この性質により，実世界の変化をActorの生成，状態更新，連携，終了として表現しながら，エッジやクラウドなど複数の計算環境で実行できます．これがActor Modelを用いるねらいです．交通監視であれば，固定カメラ，車載カメラ，観測対象を追跡する処理などをActorとして表現できます（**図1**）．

■Actor Modelだけでは解決しない課題

Actor Modelは，状態を持つ処理主体とメッセージ通信の枠組みを提供します．しかし，その時点でどのActorと連携すべきか，Actorをどこで実行すべきかまでは決めません．

まず，Actorの数や関係が変化する環境では，その時点で必要なActorを探索し，連携関係を更新する必要があります．

次に，Actorは状態を持って長時間動作するため，実行場所を運用中も見直す必要があります．他のActorの生成や終了，計算環境の負荷，通信相手が変化すれば，生成時に決めた配置が適切でなくなる場合があります．また，一部のActorが終了すると，空き資源が複数の実行環境に分散し，計算資源を効率的に利用できない場合があります．

さらに，Actorを再配置する際には状態を維持する必要があります．単純に停止して別の場所で起動し直すと，それまでの状態が失われます．また，他のActorからは，実行場所が変わっても同じActorとして呼び出せる必要があります．

そこで本技術では，Actor Modelを基本とし，その上にActor間の動的連携，運用中の動的配置，状態を維持した再配置の機能を備えた共通の実行基盤（基盤）を構築しています[(1)]．

Actorの動的連携と動的配置

■連携相手の探索

各Actorは，自身の位置，観測対

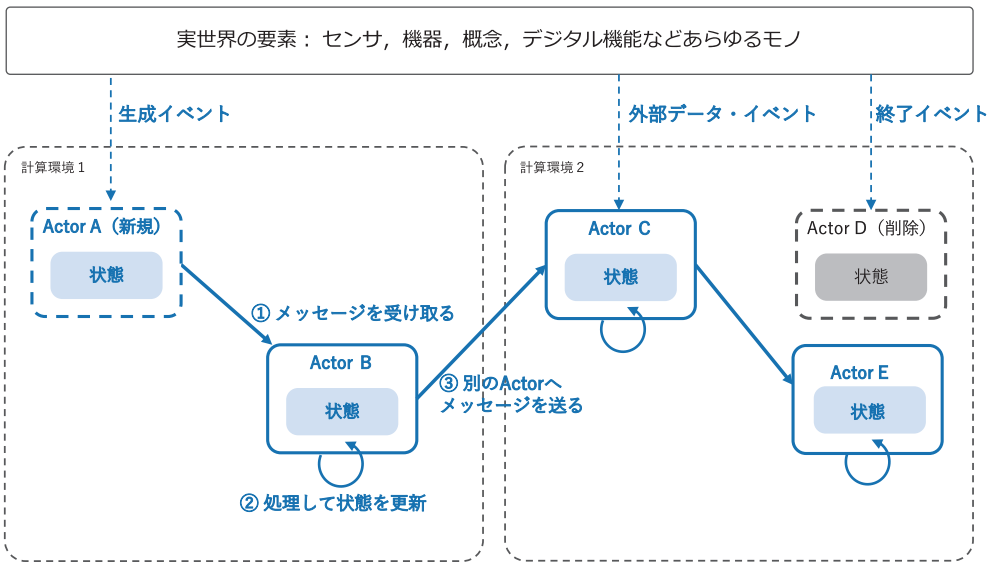

図1　Actor Modelの概念と実世界との対応

象，役割など，連携相手を選ぶための情報を基盤へ登録します．基盤はこれらの情報から，その時点で関係するActorを探索します．

本技術では，位置を格子状の領域に分割し，同じ領域に存在するActorを近傍として探索する方式を実装しています．探索方法は位置に限定せず，観測対象や機能など，サービスに応じて変更できます．これにより，すべてのActorをあらかじめ接続するのではなく，その時点で必要なActorだけで連携関係を形成できます．

Actor間の通信には，サーバのIPアドレスなどの物理的な実行場所ではなく，Actorごとに変わらない論理名を使用します．基盤は論理名と現在動作しているActorとの対応を管理するため，Actorが別の計算環境へ移動しても，他のActorは同じ論理名で呼び出せます．

■Actorの動的配置と再配置

Actorの実行場所は，生成時だけでなく運用中も見直します．配置制御機能は，各実行環境の負荷や空き資源，Actor間の通信関係などを基に実行場所を決定します．

例えば，頻繁に通信するActorを近くに配置して実行環境をまたぐ通信を抑える，負荷の高い実行環境からActorを移動する，複数の実行環境に分散したActorを集約して空き環境をつくる，といった対応が考えられます．しかし，何を重視するかはユースケース・利用状況などで変わります．よって本技術では配置方針を固定せず，低遅延化，負荷分散，リソース集約などの目的に応じて選択できる構成としています．

特に，Actorの生成と終了が繰り返されると，複数の実行環境に空き資源が分散します．例えば，1つの配置単位に10個のActorを収容でき，20個のActorが6つの配置単位に分散している場合，Actor数だけを考えれば必要な配置単位は2つです．散らばったActorの配置を変更し集約できれば，理論上は最大4つの配置単位を空にできます．

配置変更時には，移動先に同じ論理名を持つActorを生成し，移動元Actorの状態を引き継いだ後に移動元を終了します．この処理をマイグレーションと呼びます．状態移送は，集中した管理機能がすべて実行するのではなく，移動元Actorと移動先Actorが主体となって行います．論理名が示す呼び出し先を移動先へ切り替え，移動元でそれまでに処理した結果を含む状態を移動先へ引き継ぐことで，他のActorとの連携を維持しながら処理を継続します．

■動的連携と動的配置の一体制御

本技術の特徴は，Actor間の動的連携とActorの動的配置を連動させる点にあります．

Actorの連携関係が変わると，通信関係や適切な実行場所も変化します．そこで，動的連携によって得られたActor間の関係を，配置を見直す際の情報として利用します．実世界の変化によって「誰と連携するか」が変わり，その変化を「どこで実行するか」の判断にも反映する考え方です．

Actor Model自体は既存の一般的なプログラミングモデルです．本技術は，状況に応じて必要な相手と連携させ，適切な計算環境で効率良く継続動作させるために，Actorの動的連携，継続的な配置制御，状態を維持したマイグレーションなどの機能を備えています．

現在，分散実行基盤Ray[*2(2)]のActor機能を用いてプロトタイプを実装しています．論理名によるActor間通信，位置に基づく近傍Actorの探索，配置制御，状態を維持したマイグレーションなどの基本機能を実装し，Actorの実行場所を

＊2 Ray：Pythonで書かれたAI処理を複数の計算機に分散して実行するためのオープンソースの基盤で，Actorを単位とした実行機能を備えています．

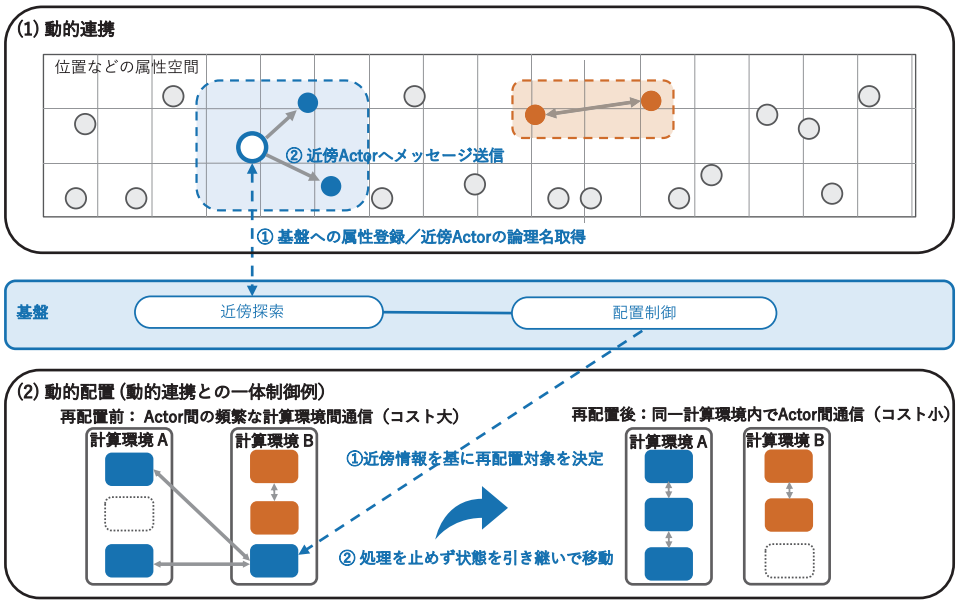

図 2　分散環境における Actor の動的連携と動的配置

変更した場合にも状態と連携を継続できることを確認しています（**図 2**）．

交通監視への適用例

本技術の特徴が活きるユースケースとして，固定カメラと車載カメラによる交通監視を説明します．このケースでは，車両や自転車のような実世界の対象の出現，移動，消失に，Actor の生成，連携先の変更，再配置，終了を対応付けることで，分散した AI が実世界の変化に追随して協調する認識処理をめざします（**図 3**）．

固定カメラ Actor が自転車を検出すると，その自転車を担当する追跡 Actor を生成します．追跡 Actor は，自転車の位置や速度，観測結果を状態として保持し，現在位置を基に周辺の車載カメラ Actor を探索します．そして，観測に適したカメラへ解析を依頼し，得られた結果から自転車の状態を更新します．

各カメラ Actor は，対応する映像を取得しやすいエッジ側の計算環境などで動作します．そのため，固定カメラ Actor，複数の車載カメラ Actor，追跡 Actor は，必ずしも同じ計算環境で実行されません．複数の計算環境に分散した Actor が，必要に応じて解析結果をやり取りしながら 1 つの認識処理を構成します．

自転車や車両が移動すると，観測に適したカメラも変わるため，追跡 Actor は連携するカメラ Actor を選び直します．また，主な通信相手や計算環境の負荷が変化した場合には，追跡 Actor などの実行場所も見直します．

この構成では，すべてのカメラ映像を 1 カ所へ集約して処理する構成とは異なり，分散した計算環境で各カメラの解析を行い，その結果だけを Actor 間でやり取りします．さらに，対象の検出や移動を契機に，その時点で認識に必要なカメラだけを連携させます．これにより，映像伝送や AI 推論を必要な範囲に絞りながら，複数の視点を用いた認識を継

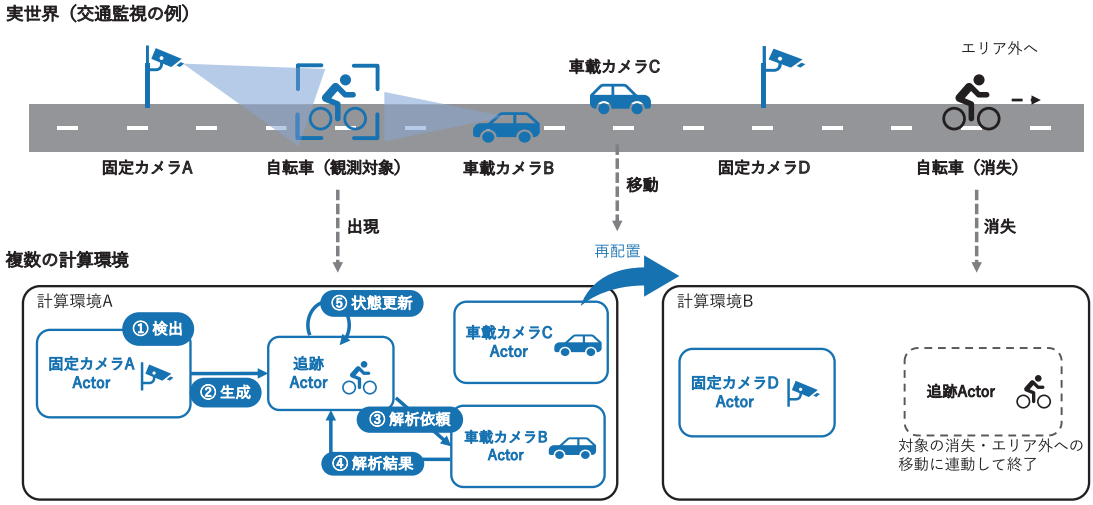

図 3　分散した固定カメラActorと車載カメラActorによる交通監視

続できます．

このように，実世界における対象の出現，移動，消失を，Actorの生成，連携先の変更，再配置，終了へ対応付けることで，分散したAIが実世界の変化に追随して協調する認識処理を実現します．

今後の展開

今後は，実世界の状況やサービス要件に応じて，多様なAI処理を安定して継続動作させるための基盤機能を高度化します．例えば，Actorが保持する状態量や処理特性，CPUやGPUなどの計算資源の違いを考慮した配置・再配置方式を検討します．また，マイグレーション時の処理継続性や，障害発生時にもサービスを継続するための運用機能についても検討を進めます．さらに，複数の処理主体が分散環境で状態を持ちながら継続的に協調する新たなユースケースとして，LLM（Large Language Model）を用いたAIエージェントやドローン・ロボットといったフィジカルAIなどへ展開していきます．

■参考文献

(1) T. Okubo, T. Fukushima, A. Sakamoto, R. Hoshino, N. Inoue, K. Yokota, and Y. Ichiyanagi: “AgentServe: Online Who-Where Adaptation for Open-World, Geo-Distributed Stateful Multi-Agent Systems,” Proc. of AAMAS 2026 (Demonstration Track), pp. 4089-4091, 2026.
(2) P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica: “Ray: A Distributed Framework for Emerging AI Applications,” Proc. of OSDI 2018, pp. 561-577, 2018.

（上段左から）大久保 淑実／坂本 啓／星野 玲那
（下段左から）横田 謙祐／福島 卓弥／外山 将司

これまでAIは，1つのモデルの性能を高めることに主眼が置かれてきました．今後は，複数のAIが状況に応じて連携し，動き続ける実世界をとらえることが重要になると考え，その土台となる基盤づくりに取り組んでいます．

◆問い合わせ先
NTTコンピューティングテクノロジーセンタ
AICP技術開発プロジェクト