原著論文 Original Paper

クラウド親和性を有する大規模社会シミュレーション基盤の設計と評価

Design and Evaluation of a Large-Scale Social Simulation Infrastructure with Cloud Affinity

平野 流 Rui Hirano
名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
ruirui@ucl.nuee.nagoya-u.ac.jp, https://ruihirano.github.io

青木 俊介 Aoki Shunsuke
（同　　上）
shunsuke@nagoya-u.jp

米澤 拓郎 Takuro Yonezawa
（同　　上）
takuro@nagoya-u.jp

河口 信夫 Nobuo Kawaguchi
名古屋大学大学院工学研究科, 名古屋大学未来社会創造機構
Graduate School of Engineering, Nagoya University and Institutes of Innovation for Fu- ture Society, Nagoya University
kawaguti@nagoya-u.jp

keywords: large-scale simulation, distributed simulation, traffic flow, multi-agent simulation, cloud computing

Summary

Cities are faced with various urban challenges such as mitigating traffic congestion, evacuation planning, and controlling the spread of infectious diseases. Multi-agent simulations are expected to be utilized to solve these problems. However, large-scale multi-agent simulations at the city level are computationally very large, making it difficult to perform them on a single computer. Currently, there are large-scale social simulation infrastructures that utilize distributed simulation technology, but they are not readily available because they require complex environment construction such as coordination on multiple computing resources, resource management, and network configuration. In this paper, we propose a large-scale social simulation infrastructure that can be easily executed in the cloud using Kubernetes / Docker, a modern container virtualization technology, and Synerex, a new distributed system. This simulation infrastructure has not only functions related to distributed simulation, such as distributed computation by spatial partitioning and dynamic resource management, but also practicality so that users can easily use it on the cloud. We have also demonstrated the simulation of the spread of infection to 100,000 people in Nagoya City, Aichi Prefecture, on the GKE cloud service provided by Google, and have shown that the execution time can be reduced to about 1/5 by increasing the number of servers.

1. は　じ　め　に

都市において渋滞緩和や災害時の避難誘導計画の策定，感染拡大の抑制は重要である．2021 年の東京オリンピックや 2025 年の大阪万博のような大規模なイベントでは，局所的かつ突発的な交通渋滞や人混みが頻繁に起こると予想されており，これらの問題に対する早急な対応が必要不可欠である．また，2019 年末におきた Covid19 によるパンデミックの状況では，いかに感染拡大を抑制できるかが重要な問題となっている．

このような都市課題を解決するために，マルチエージェントシミュレーション (以下，MAS)[**?**] による社会シミュレーションの活用が期待されている．MAS とは，人や車などを模した複数のエージェントを自律的に行動させ，相互作用によって起こる現象を分析するシミュレーション技術である．MAS は個々のエージェントに行動規則を持たせられるため，異なる意思をもつ人間が相互作用するような現実社会のシミュレーションに適しており，人流や交通流シミュレータなど MAS を活用した様々なシミュレータが存在する．

MAS の課題として，全てのエージェントの行動を一つ一つ計算するため，大規模化に伴い計算量が莫大に増加するという課題が挙げられる．都市レベルの超大規模な社会シミュレーションを想定すると，計算リソースを単一のコンピュータに収めることは難しい．そのため，大規模な社会シミュレーションの実行には，シミュレーションによって生じる計算を複数の計算資源へ分散し，お互いが協調しながら完結させる，分散シミュレーションが必要不可欠である．

しかしながら，MAS を利用した既存のシミュレータは

分散環境における実行を想定していないものが多く，大規模化が非常に難しい．また，複数の計算資源を連携するには，OS やネットワーク，リソース容量など，計算資源に合わせたアプリケーションの配置や複雑な設定が必要であり，既存の多くの社会シミュレーション基盤は誰もが容易に利用可能な環境にはなっていない．

近年，クラウドコンピューティング分野やその周辺技術の発達により，実行環境による影響の抑制や複数資源間での一貫したリソース共有が可能な Kubernetes / Docker [?] というコンテナ仮想化技術が現れた．Kubernetes / Docker は複数の計算資源の管理や動的なプロビジョニングをユーザーに提供する．Amazon や Google などの大手ベンダーも EKS(Elastic Kubernetes Service) や GKE(Google Kubernetes Engine) を提供し，サポートしている．

我々は人流や交通流，災害などを対象とした既存のシミュレータを基盤上に載せることで今まで困難だった大規模化を可能にし，複雑な環境構築を必要とせずに都市レベルの大規模な社会シミュレーションを行えるようなシミュレーション基盤の実現を目指している．

本研究では，モダンなコンテナ仮想化技術である Kubernetes / Docker と新たな分散システムである Synerex[?] を用いて，EKS や GKE などクラウド上での実行が容易な大規模社会シミュレーション基盤を提案する．本シミュレーション基盤は，空間分割による計算量の分散や動的なリソース管理といった，分散シミュレーションに関わる機能だけでなく，利用者がクラウド上で容易に使えるような実用性も有している．また，愛知県名古屋市を想定した 10 万人規模の感染拡大シミュレーションの動作実証を行い，本基盤を用いたクラウド上での社会シミュレーションの有効性を示した．

本研究による貢献は次の 3 つである．

(1) 複数の計算資源を動的に拡張可能な大規模社会シミュレーション基盤を設計・構築したこと
(2) モダンなコンテナオーケストレーションツールである Kubernetes を用いて，クラウド上で複数の計算資源を柔軟に連携させたシミュレーション実行が可能となる仕組みを有していること
(3) 既存のクラウドサービスを通して，10 万規模の大規模社会シミュレーションの動作を実証したこと

本論文の構成は次に示す通りである．まず，2 章で既存の交通流シミュレータや分散シミュレーション技術，クラウドコンピューティング技術に関する関連研究を紹介し，本論文で扱う課題を明確化する．3 章では，クラウドサービスとの親和性を持つ社会シミュレーション基盤のアーキテクチャについて説明する．4 章で本シミュレーション基盤によるクラウド上での動作実証について述べ，5 章でまとめと今後の課題について述べる．

2. 関 連 研 究

MAS を用いた既存の交通流シミュレータとして，高度交通システム (ITS) の分野でよく用いられる SUMO (Simulation of Urban MObility) [?] が挙げられる．SUMO は大規模な道路交通を制御するために開発されたオープンソースの交通流シミュレータである．他にも Carla [?] に代表される自動運転システムを交えた交通流シミュレータも多く存在している [?, ?]．しかしながら，これらのシミュレータにはシミュレーションエリアの制限やエージェント数の制限が存在し，大規模なシミュレーションを実行するのは容易ではない．

MAS の大規模化に向けた分散シミュレーション基盤に関する研究は 1990 年代頃から行われてきた．分散シミュレーション基盤を構築する規約として HLA(High Lebel Architecture)[?] が挙げられる．HLA は 1990 年代に米国防省によって提唱された，シミュレータ間の情報共有を目的とした標準規格であり，Run Time Interface という実行基盤を連携することで，複数のシミュレータを用いた分散シミュレーションを可能にする．しかしながら，HLA は異種シミュレータの連携に焦点を当てているものの，シミュレーションの大規模化に関しては焦点を当てていない．また，鈴村らは並列分散プログラミング言語 X10 を用いた大規模交通流シミュレーション「XAXIS」を開発し，スーパーコンピューター TSUBAME2.0 上で 1 千規模級の大規模交通シミュレーションの高速な実行を実現した [?].

また，2010 年代のコンテナ型仮想化技術の急速な進化により，計算リソースを柔軟に適用できるクラウド環境で MAS を実行する研究が行われるようになった．実際に，Taria は文献 [?] において，クラウドコンピューティングのモデルとアーキテクチャ，並列および分散アプリケーションでのそれらの使用について説明し，クラウドコンピューティングとマルチエージェントシステムには類似点や共通の課題が多く非常に親和性があると述べている．また，Ralha らはクラウドコンピューティングプラットフォーム上でリソース管理を行う「MAS-Cloud」というツールを開発・評価した [?]．このツールはリアルタイムアプリケーションにおけるリソース管理であり，社会シミュレーションに関する設計までは考慮していない．

このように，分散シミュレーションやクラウド上の社会シミュレーションに関する研究は行われているものの，複数の計算資源上での社会シミュレーションには，異なる実行環境での動作やネットワーク設定，リソース管理など，複雑な設定が必要となり実行コストが高いという問題がある．そのため，既存の社会シミュレーション基盤の多くは誰もが容易に利用できる環境にはなっていない．本研究では，大規模性能やシミュレーション効率といった機能性だけでなく，Kubernetes / Docker と分散システム Synerex を用いて，利用容易さやクラウドへの親

和性など実用性を兼ね備えた社会シミュレーション基盤の設計と構築を行う．

3. 大規模社会シミュレーション基盤の設計

社会シミュレーションの大規模化を実現するためには，シミュレーション自体の分散システムの他に，柔軟な計算リソースの管理と，環境依存の少ないクラウドとの親和性が必要不可欠である．我々はさらに利用者がリソースや計算資源を考えることなく利用できることが理想であると考える．そのため，利用障壁を下げることも重要な要件である．初めに，本シミュレーション基盤が使用している分散システム Synerex の概要を述べる．次に，大規模化におけるシミュレーション設計とクラウド上でのリソース管理におけるアーキテクチャを述べ，さらに利用容易さに関する設計指針について説明する．

3·1 需給情報交換システム Synerex の概要

Synerex[*1][?] は社会のサービス提供者と利用者間における需給の効率的な交換を可能にする需給交換基盤である．Synerex の概要と利用例を図 1 に示す．Synerex はデータ交換基盤となる Synerex Server とデータを交換する主体となるプロバイダによって構成される．プロバイダは利用するアプリケーションによって自由にカスタマイズできる．例えば，サービス提供者プロバイダやサービス利用者プロバイダ，データ提供プロバイダなど，利用したい目的に合わせた開発が可能である．

Synerex Server は各プロバイダに対して需給交換に適した独自の API を提供する．API には「NotifyDemand」，「ProposeSupply」，「SelectSupply」，「Confirm」の 4 種類が存在し，それぞれ需要の通知，供給の提案，供給の選択，確認という需給において必要な機能を提供している．また，内部システムには，Google によって開発されている gRPC と Go 言語を用いている．gRPC のデータ内容の記述による一貫性のある API 定義と，Go 言語が有する並列実行によるリアルタイムな通信により効率的な需給交換を実現している．

加えて，Synerex は Synerex Server 自体を分散可能な柔軟性の高い分散システムである．図 1 のようにゲートウェイとなるプロバイダの接続により複数の Synerex Server を協調できる．これにより，データ交換基盤に集中しやすい負荷の分散が可能となる．これらの特徴はクラウドを利用した分散シミュレーションとの相性が良いため，本シミュレーション基盤では複数資源上での連携に加えて，動的なリソース管理に利用している．

3·2 Master と Executor によるシミュレーション構成

図 2 に本シミュレーション基盤の全体構成を示す．本シミュレーション基盤は Master レイヤと Execution レイヤ

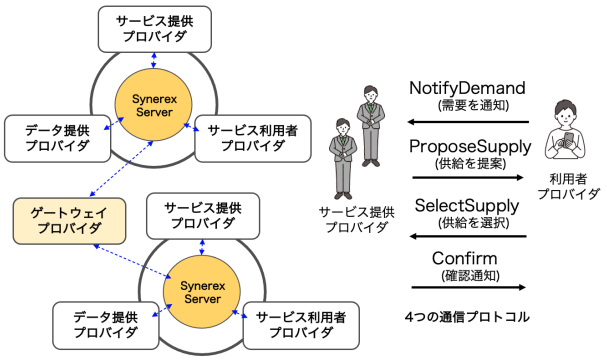

図 1 Synerex の概要と利用例

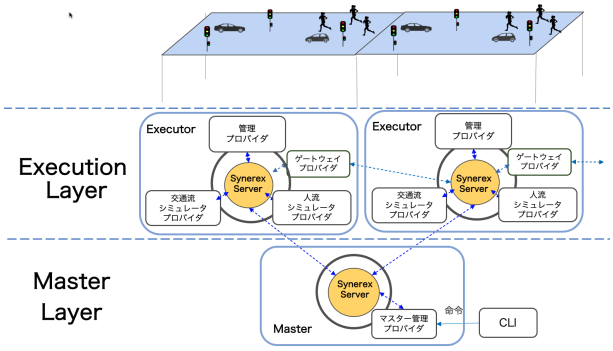

図 2 全体構成図

という二つのレイヤで役割を分けている．Execution レイヤには，それぞれの計算資源上に分散された Executor が存在する．各 Executor は担当するエリアを保持しており，そのエリアにおけるエージェントの計算や他 Executor とのエージェント共有など，実際のシミュレーション計算に関わる役割を担う．Master レイヤには，時刻同期やシナリオ設定，Executor 管理など，シミュレーション全体の一貫性を保つ役割を有する Master が存在する．また，本シミュレーション基盤では，各 Executor 間でエージェント共有が行われるものの，Master には通信負荷の大きいエージェント情報は渡さないようにすることで，Master への負荷の集中を抑制している．

3·3 空間分割による計算量の分散

MAS では個々のエージェントを全て計算するため，シミュレーションが大規模になるほど計算量が増加する．そのため，大規模なシミュレーションはリソースの限られた単一のコンピュータ上での実行が難しくなる．本シミュレーション基盤は複数の計算資源上で一貫性を保ちつつ動作を行う分散シミュレーションを前提としている．

本シミュレーション基盤では，シミュレーションエリアの空間分割を行う．図 3 のようにエリアやエージェント毎に空間を分割し，各オブジェクトが通信を行いなが

*1 GitHub 「Synerex Project」. https://github.com/synerex.

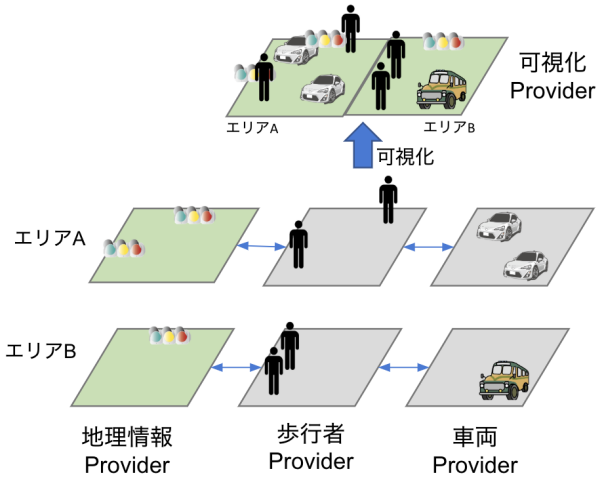

図 3 シミュレーションエリアの空間分割

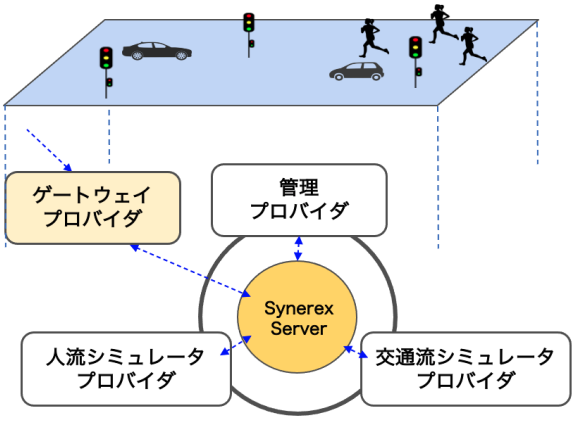

図 4 Executor の概要図

ら協調することで，シミュレーション計算の並列実行による高速化を試みている．

また，個々のオブジェクトを独自に通信させると，通信がメッシュ構造になり複雑になる．そのため，通信による情報交換には Synerex を用いている．図 4 に一つの空間における Executor の概要図を示す．Executor 内には，車両のシミュレーションを実行する交通流シミュレータプロバイダや，歩行者のシミュレーションを実行する人流シミュレータプロバイダなどシミュレータ毎のプロバイダが存在する．管理プロバイダは時刻同期やシミュレーションのサイクル促進など Executor 内の管理を行う．さらに，ゲートウェイプロバイダによって，隣接しているエリアとの情報交換を行い協調する．

3·4 重複エリアによる通信量の抑制と一貫性の保持

シミュレーションエリアの空間分割は計算量を分散できるという利点がある一方，隣接エリアのエージェントとの相互作用を考慮するため，エージェント情報の取得

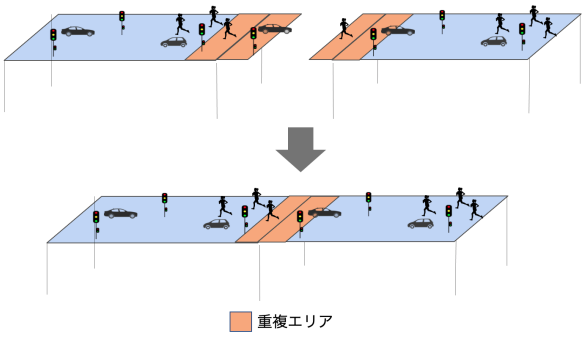

図 5 重複エリアの概要図

による通信が発生する．その際の通信数や通信量を可能な限り削減するため，本設計では重複エリアという概念を用いている．重複エリアとは，図 5 のように隣接するエリア境界でのエージェントの相互作用を考慮するために用いられる領域であり，重複エリアにいるエージェントは隣接する両エリアで保持される．

図 6 に，シミュレーションの 1 サイクル実行時の通信フローを示す．各シミュレータプロバイダは管理プロバイダからサイクル実行命令を受けた時に以下のような順番でサイクルを進める．

(1) 管理プロバイダからサイクル実行命令を受け取る
(2) 同じエリアにいる異種エージェントプロバイダからエージェント情報を取得する
(3) 取得した同じエリアのエージェント情報を用いてシミュレーションを 1 サイクル実行する
(4) 隣接しているエリアの同じ種類のエージェントプロバイダから重複エリアに存在するエージェント情報を取得する
(5) 取得した隣接エージェント情報を用いて重複エリアを更新する

各エリアは同じエリアにいる異種エージェントと隣接したエリアにいる同種のエージェントの交換のみを行う．そのため一つの Executor の通信数はサイクルによってほとんど変わらない．また，通信量は重複エリアを含めた自エリアにいるエージェント数のみに影響を受けるが，自エリアの外側にいるエージェントからは影響を受けない．このような仕組みによって通信のローカリティを保っている．

3·5 シミュレーション実行に関わる通信フロー

シミュレーション実行やエリアの拡張には，Master レイヤと必要なエリアに応じた Executor を起動する必要がある．ここでエリアの拡張を容易にするため，Executor による Master への参加登録の仕組みを用いている．図 7 は，エリア拡張時に 2 つの Executor が Master へ参加登録を行う通信フローである．まず、Executor1 が起動時に Master に対して参加登録申請を行う．それに対して Master が登

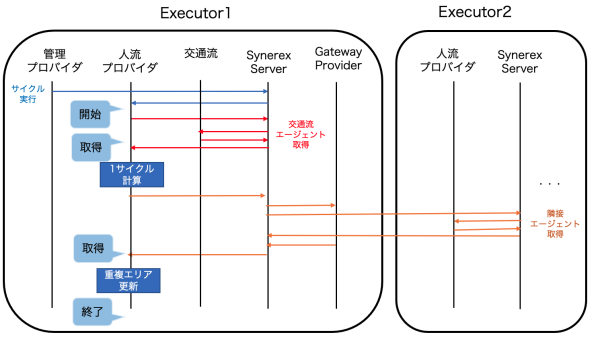

図 6　時刻同期フロー

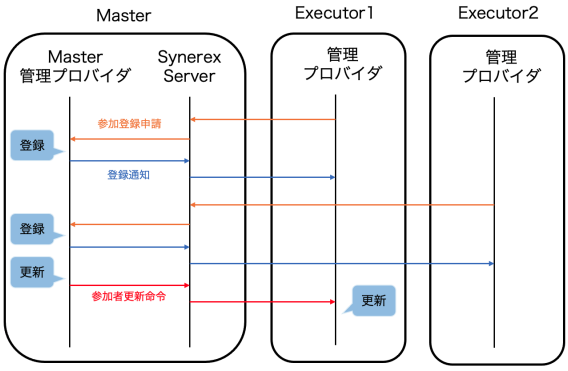

図 7　参加登録フロー

録を行い完了通知を返す．さらに新しい Executor2 が起動した場合，Master は登録後全ての Executor に参加者更新命令を行う．このような通信フローにより，Executor と Master は常に新しい参加者を共有できるため，シミュレーション実行中であっても矛盾なくエリアの拡張が行える．

3·6　Synerex を利用したリソース管理手法

複数の計算資源上での分散シミュレーションでは，エリア内に存在するエージェントの密度により計算量が動的に変化する．そのため，異なるリソースを持った計算資源を動的に分配するリソース管理が重要である．本シミュレーション基盤では，Synerex が有する独自の通信プロトコル「NotifyDemand」,「ProposeSupply」,「SelectSupply」,「Confirm」の 4 種類の API を用いて，リソース管理を行っている．図 8 にリソース管理による通信フローを示す．NotifyDemand は需要の通知を行うプロトコルであり，Executor 内のリソースが足りない場合に周辺エリアの Executor のリソースを求めることに用いている．ProposeSupply は需要に対して供給を行うためのプロトコルであり，NotifyDemand を受けた Executor が自身のリソースが存在する場合に供給可能の通知を行うために使用する．SelectSupply は受けた供給に対して利用するものを選択し通知するプロトコルで，利用するリソース

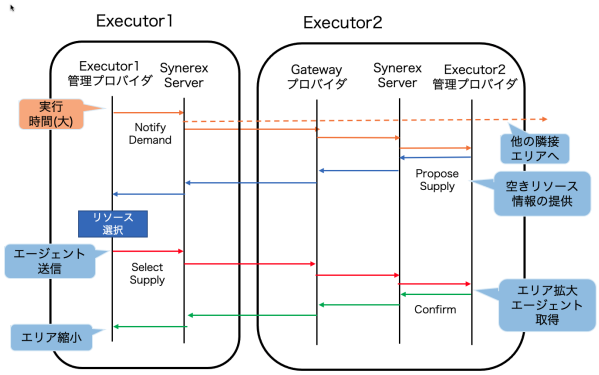

図 8　リソース管理フロー

とその容量の選択と供給先への通知に使われる．Confirm は供給先の確認を通知するプロトコルで，供給元から需要先へ必要なリソースの提供を行う．また，このフローで行われるリソースの提供は，負荷の大きい Executor のエリアの一部を供給元の Executor に分配し，エリアの大きさを変動させることによって行う．このような一定のフローを各 Executor で行うことにより，柔軟性を持った動的なリソース管理が可能となる．

3·7　Kubernetes / Docker によるリソース管理

従来の社会シミュレーションでは，複数の計算資源を用いて分散協調する場合，各計算資源を接続するネットワーク設定や OS に合わせたソフトウェアの起動，計算資源の規模調整など多くの課題があり，容易に利用できる環境ではなかった．このような課題はクラウドコンピューティング分野でも共通の課題であり，これらを解決するために Docker や Kubernetes といったコンテナ仮想化技術，コンテナオーケストレーション技術が活用されている．

Docker とは，サーバのカーネルを利用して実行環境を他のプロセスから隔離するコンテナ型のアプリケーション実行環境である．Docker を利用することで，物理マシン上の環境変化による問題を減らすことが可能になる．Kubernetes はコンテナ化されたアプリケーションをクラスタ上でデプロイやスケーリングを行うシステムである．Kubernetes では Pod と呼ばれる仮想的なサーバを作成し，設定ファイルにしたがって複数の物理マシン上にリソースを最適化するように設置する．また，高度な可用性を備えており，Pod が停止した場合に自動で再起動するなど Pod の死活監視も可能である．

このように Docker/Kubernetes は高可用性や拡張性，保守性を有しているため，多くのサービスのインフラ環境として活用されている．また，Amazon や Google など大手ベンダーにも EKS や GKE といったクラウドサービスとしてサポートされており，コンテナ仮想化およびオーケストレーションツールとしてはデファクトスタンダードになっている．

本シミュレーション基盤では，Docker / Kubernetes を

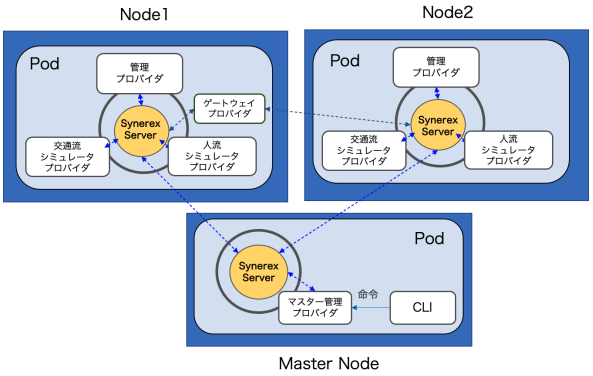

図 9 Kubernetes Pod の図

活用し，図 9 のように Master Pod と複数の Executor Pod を Kubernetes クラスタ上で分散している．また，3•6 節で述べた，Synerex を用いたアプリケーション内でのリソース管理の他に，Kubernetes 独自のオートスケーリング機能を用いたリソース管理も行っている．Kubernetes では Pod 自体のリソースを増加させる垂直オートスケーリングと，Pod のリソース利用率が設定した閾値を超えた場合に Pod を増やす水平オートスケーリング機能があり，これらの機能を活用してアプリケーション外からの動的なリソース管理を行っている．

3·8 本シミュレーション基盤の利用

本シミュレーション基盤はオープンソースとして GitHub 上で公開しながら日々アップデートをしている *2．現在，Master と Executor との連携や重複エリアの実装，Kubernetes の設定ファイル自動生成機能など基本的な要件は実装済みであり，簡単なシミュレーションをクラウド上で大規模に動かすことに成功している．今後はリソースを動的に管理する機能の実装や，ユーザの利用しやすいシナリオ構築機能などを実装し追加を行う予定である．

以下に，本シミュレーション基盤の利用手順を示す．

(1) 本シミュレーション基盤をローカルにインストールする
(2) 使用するシミュレータのプラグインを作成する
(3) シミュレーションのシナリオ設定を記述する
(4) Kubernetes 用の設定ファイルを生成する
(5) Kubernetes 対応のクラウド上で設定ファイルからアプリケーションを起動する

本シミュレーション基盤では，使用するシミュレータに応じて独自のプラグインを作成する必要がある．プラグインの作成にはある程度時間がかかるが，将来的には開発するためのテンプレートの提供や，一度作られたプラグインをその他の利用者が共有できるようなプラットフォームの提供による対応を考えている．

また，Kubernetes の設定ファイルは，利用者が自分で作成する必要はなく，シミュレーションの規模やエージェント数を記述したシナリオ設定に応じて自動で生成される．これにより，利用者は複数の計算資源を意識することなく，クラウド上でアプリケーションを起動するだけで，容易に大規模な社会シミュレーションを実行できる．

4. 大規模感染シミュレーションの実装とクラウド上での検証

4·1 感染シミュレーションの実装

本シミュレーション基盤上に感染シミュレーションを実装するにあたり，一般的な感染モデルである SIR モデル [?] を利用し，感染シミュレータを作成した．SIR モデルは感染症の流行過程を表したモデル方程式である．頭文字がそれぞれ感受性保持者 (Susceptible)，感染者 (Infected)，免疫保持者 (Recovered) を表しており，感受性保持者が感染者に接触するとある割合で感染し，その後一定期間経過すると回復するという流れとなる．

また，エージェントは人のみとし，行動アルゴリズムとして RVO2[?] を利用した．RVO2 は Velocity Obstacles という人同士が接触しない範囲を定義することで，人の回避行動を表現するアルゴリズムである．さらに，シミュレーション結果の可視化には「Harmoware-VIS」と呼ばれる Deck.gl ベースの可視化ライブラリ [?] を利用した．図 10 に HarmowareVIS を用いて道路ネットワーク上にエージェントの一部を可視化したものを示す．緑色のエージェントが感受性保持者を示しており，赤色のエージェントが感染者を示している．シミュレーションエリアは愛知県名古屋市を想定しており，国土交通省の G 空間情報センター *3がオープンデータとして提供している名古屋市歩行者ネットワーク情報を利用して，そのネットワーク上に歩行者を設置した．

4·2 検 証 環 境

本シミュレーション基盤を用いてクラウド上における社会シミュレーションの動作実証を行った．本実験の目的は現実の状況を想定したモデルでクラウド上での大規模な社会シミュレーションが可能かどうかを検証すること，サーバー台数の増加によるシミュレーション実行時間の変化を明らかにすることである．

本実験で使用したクラウドサービスは，Google の提供する GKE(Googke Kubernetes Engine) である．今回の実験では，表 1 のような性能を持つ仮想サーバを複数台使用した．本実験では名古屋市の人口およそ 230 万人の約 5%にあたる 10 万人のエージェントの 1 時間の動きを想定し，感染流行過程をシミュレーションした．今回の実験では，感染者を全体の 25%とし，感染者の半径 2m 以内にいると 20%の感染確率で感染し，1 日後に回復する

*2 GitHub 「Flow」. https://github.com/RuiHirano/flow_beta.

*3 G 空間情報センター 「歩行空間ネットワークデータ等」. https://www.geospatial.jp/ckan/dataset/0401.

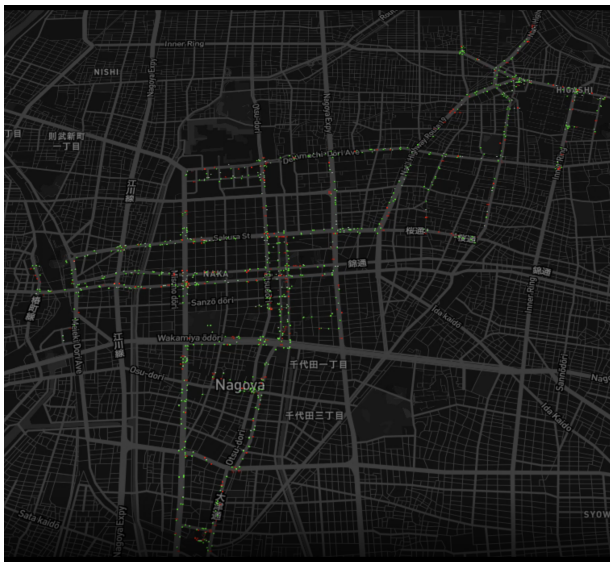

図 10　可視化の図

表 1　使用した仮想サーバの性能

使用した機材等	詳細
マシンタイプ	N1-Standard-1
CPU 性能	Intel Xeon Silver 4214，2.20GHz
使用 CPU コア数	1Core
使用メモリ	2GB

というパラメーターを設定した．また，サーバー台数を増やし，感染拡大シミュレーションの実行にかかる時間を計測した．

4·3 検 証 結 果

本実験のシミュレーション結果を図 11 に示す．SIR モデルの回復にかかる時間を 1 日と設定したため，回復者数は常に 0 となっている．感染者数は，10 万人のうち 25%を占める 25000 人が感染から始まり，約 1900 秒経過した時点で感染者が感受性保持者を上回った．それ以降，感染者数が増加する速度は弱まるも単調増加し，1 時間後には約 56000 人まで増加した．

次に，サーバー台数とシミュレーションの実行時間の関係を表 2 と図 12 に示す．サーバー台数が 1 台の時は計算時間が 3520ms で最大だが，エージェントの交換が起きないため通信時間は 14ms と最小となり，シミュレーション実行時間は 3.56 時間となった．サーバー 4 台の時は，計算時間は約 2/5 程度まで削減できているが，エージェントの通信が発生するため通信時間が 2682ms と最大になり，シミュレーション実行時間は 3.78 時間と 1 台と比べて増加した．それ以降はサーバー台数が増加するにつれて計算時間と通信時間は減少し，サーバー台数 25 台使用時ではシミュレーションの実行時間は 0.67 時間となり，1 台での実行時間の約 1/5 程度に抑えることができた．

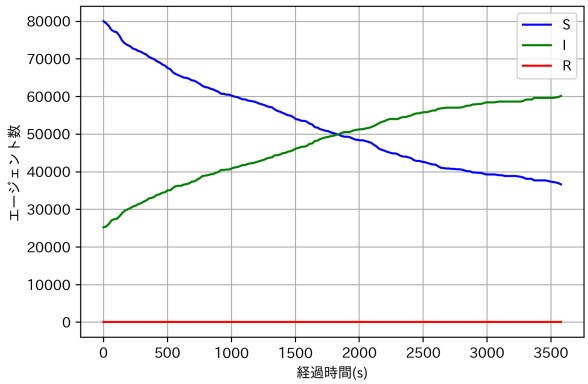

図 11　シミュレーション結果

表 2　サーバー台数と実行時間の関係

サーバー台数	1	4	9	16	25
計算時間 (ms)	3520	1374	382	200	143
通信時間 (ms)	14	2682	1217	908	513
1 周期時間 (ms)	3556	3785	1613	1237	681
実行時間 (h)	3.56	3.78	1.61	1.23	0.67

4·4 考　　察

本実験により，GCP 上での 10 万人規模の感染シミュレーションが可能であることを実証できた．GCP 上での実行ファイルは自動で生成されるため，実行環境の構築は 1 時間もかからず行えた．

またサーバー台数と実行時間の関係から，サーバーを複数台使用する場合，台数が増えるほど計算時間，通信時間ともに減少している．これは，エリアを分割するほど一つのエリアが担当するエージェント数が減少し，計算量が減少すると同時に隣接エリアとの通信量も減少していくためであると考えられる．これにより，空間分割とサーバー分散による通信のローカリティを担保できていることがわかった．

一方で，実行時間の減少幅は台数が増えるほど小さくなっており，利用台数を増加させても実行時間の変化が小さく十分な効果が得られない可能性があることがわかった．今回の実験では，計算時間と通信時間の減少幅はサーバー台数の増加に伴い小さくなっている．そのため，シミュレーション実行時間と台数拡張に伴うコストを見極め，利用者の費用対効果も考慮してサーバー台数を拡張する必要があるだろう．

また，人口分布の偏りによる影響への対応も今後の課題である．今回の実験ではシミュレーションエリアを均等に分割したため，エージェントの多いエリアと少ないエリアが存在していた．Master の時刻同期では，一番遅い Executor に合わせるため，このようなエージェントの偏

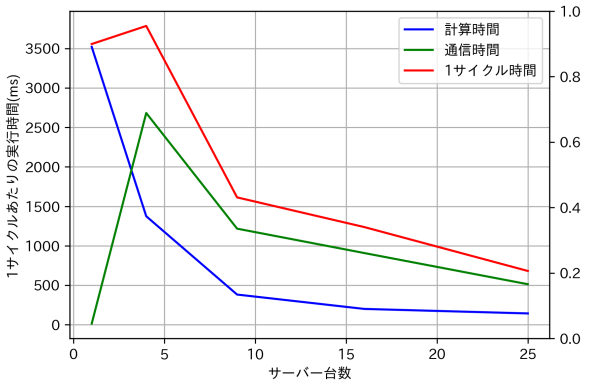

図 12　サーバー台数と実行時間の関係

りは実行時間に大きな影響を及ぼす．この問題には，各 Executor が担当するエリアをエージェント数によって動的に変更し，エリア毎のエージェント数を平滑化すれば対応できると考えている．

5.　まとめと今後の課題

本論文では，クラウド親和性を有する大規模な社会シミュレーション基盤の設計を提案した．シミュレーションエリアの空間分割と重複エリアによる分散シミュレーションに関する設計と，Kubernetes / Docker と Synerex によるクラウド実行を容易にするための設計について考案し，実装を行った．また，Google のクラウドサービスである GKE 上で本シミュレーション基盤の動作実証を行った．結果として，10 万規模の感染シミュレーションの実行が可能であり，サーバー台数を増やすことで実行時間を短縮できることを示した．

今後の取り組みとしては次の 2 つが挙げられる．1 つ目は，リソースの動的な管理機能を実装し，機能追加を行うことである．現状，自動で生成された Kubernetes 設定ファイルを利用して，クラスタ上での最適な初期配置を自動化することには成功している．しかしながら，実行途中にエージェントが増えたり，エリアが拡大するなどのアプリケーション内部で起こる動的なリソース増加には対応できていないため，今後の課題となる．2 つ目は，クラウドサービス上での性能評価を行うことである．これまでの取り組みで，クラウド上で 10 万規模のシミュレーションの動作実証と実行時間の変化を検証できた．一方で実行効率や台数効果などの性能についてはまだ評価できていない．今後追加で性能評価を行い，本シミュレーション基盤の有効性や実用性を示していく必要がある．

謝　　辞

本研究は，JST CREST JPMJCR1882，NICT 委託研究，総務省 SCOPE（受付番号 191506001）委託，JST OPERA（JPMJOP1612）の支援を受けたものです．

著　者　紹　介

平野　　流（学生会員）

2019 年名古屋大学工学部電気電子・情報工学科卒業を経て，同大学博士前期課程学生．主にエージェントシミュレーション，サイバーフィジカルシステムに関する研究に従事

青木　俊介（正会員）

2020 年カーネギーメロン大学 計算機工学科 (Electrical and Computer Engi-neering) PhD 取得．船井情報科学振興財団奨学生．2020 年 11 月より名古屋大学未来社会創造機構特任助教．自動運転システム，サイバーフィジカルシステムに関する研究に従事．

米澤　拓郎（正会員）

2010 年慶應義塾大学大学院政策・メディア研究科後期課程博士号取得後，同大学院特任助教，特任講師，特任准教授を経て，2019 年より名古屋大学大学院工学研究科准教授．主に，ユビキタスコンピューティングシステム，ヒューマンコンピュータインタラクション，センサネットワーク等の研究に従事．ACM，各会員．

河口　信夫（正会員）

1990 年名古屋大学工学部電気電子工学科卒業．1995 年同大学大学院工学研究科情報工学専攻博士課程満了．同年同大学工学部助手，同大学講師，准教授を経て，2009 年より同大学大学院研究科教授．NPO 位置推定サービス研究機構 Lisra 代表理事．モバイルコミュニケーション，ユビキタスコンピューティング，行動センシングの研究に従事．博士（工学）．ACM，IEEE，人工知能学会，日本ソフトウェア科学会，電子情報通信学会，日本音響学会各会員．