情報処理学会論文誌 Vol.62 No.2 548–559 (Feb. 2021)

大規模なインタラクティブシミュレーションのためのデータの階層化と拡散を特徴とする分散システム

山岡 久俊[1,2,a)] 蔡 東生[1,b)]

受付日 2020年4月30日, 採録日 2020年11月5日

概要：広大なプレイフィールドを持つオンラインゲームや，広域な交通渋滞予測シミュレーションへの応用を目的としたインタラクティブシミュレーション基盤技術を提案，実装した．本基盤は格子状にネットワーク接続した複数のノードによってシミュレーションサーバを構成する分散システムであり，各ノードは自らの担当するシミュレーション空間区分の時間発展計算と，クライアントからの要求に応じたデータの提供と更新を行う．また各ノードは保持するデータを階層化して近接ノードに拡散し，ノードネットワーク上にシミュレーション空間の可視化に適したデータ分布を生み出す．これらにより，これまではスケーラビリティの確保が難しかったインタラクティブシミュレーションを大規模化できる．実装した分散システムを最大 100 台の計算ノードで動かして評価し，シミュレーション空間の拡張に対してサーバノードを増やすことで線形にスケールアウトできることを示した．また，多数のクライアントがシミュレーション空間に対してアクセスする状況下での負荷傾向を，従来方式とあわせてモデル化し，実験を含めた比較検証を行った．

キーワード：分散システム，分散データベース，分散共有メモリ，インタラクティブシミュレーション

Distributed System with Hierarchical Data and Diffusing for Large-scale Interactive Simulation

Hisatoshi Yamaoka[1,2,a)] Dongsheng Cai[1,b)]

Received: April 30, 2020, Accepted: November 5, 2020

Abstract: This paper presents a distributed system for large-scale interactive simulation platform, which is applicable to online games with huge play fields and simulations of urban mobility traffic prediction, etc. Our proposed platform is a distributed system in which a simulation server is composed of multiple nodes connected in a grid network. Each node calculates the time evolution of its own simulation space, and provides and updates data according to the request from the client. In addition, each node hierarchizes the stored data, and diffuse them to neighbor nodes, creating a data distribution suitable for visualizing the simulation space on this node network. These mechanisms enable large-scale interactive simulations that have not been realized. We implemented our distributed system and evaluated its performance by running up to 100 computation nodes. Our results show that it can be scaled out linearly expanding the simulation space. The load tendency under the condition that many clients access the simulation space was modeled together with the conventional method, and the comparative verification including the experiment was performed.

Keywords: distributed system, distributed database, distributed shared memory, interactive simulation, MMORPG

[1] 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] 富士通研究所サイバーフィジカル融合基盤プロジェクト
Fujitsu Laboratories Limited, Cyber Physical Fusion Platform Project, Kawasaki, Kanagawa 211–8588, Japan
[a)] S1930167@s.tsukuba.ac.jp
[b)] cai@cs.tsukuba.ac.jp

1. はじめに

自然現象が模擬的に再現された仮想世界の中で，プレイヤが地形生成や探索，構造物の構築を楽しむサンドボックス型のゲームが台頭している．Minecraft^{TM} [*1]は 1 億人を超えるユーザを持ち，科学・プログラミング教育などの用途にも用いられている代表的なサンドボックスゲームである [1]．Minecraft^{TM} は仮想世界を構成する「草」や「炎」，「土」といった種別を持つブロックを，近接ブロックとの相互作用に基づいて更新し続けることで自然現象を簡易的にシミュレートする．しかし，メモリと CPU のリソースに制約があるため，この更新処理は仮想世界上のプレイヤの存在する位置の周辺領域のみに限定され，プレイヤから離れた場所の時間進行は表示されない．他のサンドボックス型ゲームでも同様に，計算機リソースの制約から，シミュレーションはプレイヤの周辺や視界の範囲内といった特定領域に限定して実行される場合が多い．しかし，もし仮想世界全体にわたってシミュレーションを実行し，なおかつその仮想世界に対して多数のプレイヤがインタラクティブに働きかけを行うことを可能とするシミュレーション基盤があれば，これまでにない規模の広さで仮想世界とプレイヤの相互ダイナミクスを生むことができ，ゲームとしての魅力向上に大きく貢献できる．また，都市計画や環境対策などのための実験・検証ツールとして，ゲームを超えた枠組みでの応用も期待できる．このようなシミュレーション基盤のコンセプトを**図 1** に示す．また，このシミュレーション基盤に求められる課題と要件を次にあげる．

(1) 大量の空間データを保持できる

仮想世界を，1 辺の長さを 1 m と設定した 2 次元格子で表現した場合，地球と同等の広さを実現するためには約 5.101×10^{14} 個の格子が必要となる．仮に 1 格子あたりのデータサイズを 2 バイトとすると，合計データサイズは約 1.02 ペタバイトとなり，この規模のデータを保持できる必要がある．ただし，仮想世界全体が一様な格子で現されるため，場所によるデータ量の偏りは存在しない．

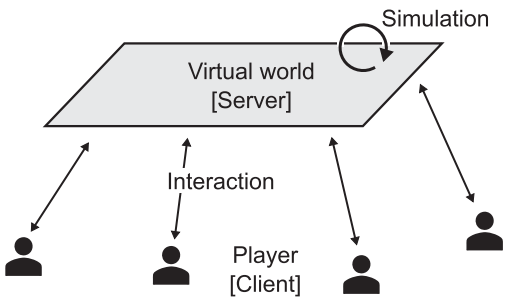

図 1 インタラクティブシミュレーション基盤のコンセプト
Fig. 1 Concept for a interactive simulation platform.

[*1] https://www.minecraft.net/

(2) 大量の格子演算を継続実行できる

仮想世界を構成する大量の格子に対する時間発展シミュレーションを遅延なく継続処理できる．インタラクティブ性を出すためには，すべての格子について遅くとも数秒ごとに近傍格子との相互作用を計算し，格子の値を更新し続ける必要がある．なおこの処理は，すべての格子に対して共通の規則を適用して実行し続けるため，場所や時間による演算負荷のホットスポットは存在しない．

(3) クライアントからの要求に遅延なく応答できる

クライアントからの格子の読み書き要求に遅延なく応答できる必要がある．クライアントの数はアプリケーションによって異なるが格子 1 辺を 1 m と設定した場合で最低でも 1 平方キロメートルあたり 1,000 クライアントからのリクエストに応えられる必要がある．なお，各クライアントは時々刻々と変化する格子状況をリアルタイムに可視化するため，視界に含まれる領域の格子データを毎秒リクエストすることを想定している．

(4) 可視化スケールを柔軟に変更できる

プレイヤが仮想世界全体の状況を俯瞰的に把握するため，仮想世界のエリアを広域的に可視化できること．また，インタラクティブな操作のために，広域表示と特定エリアに絞った詳細表示との遷移をシームレスに行えること．

(5) 低い運用コストでサービスを継続できる

基盤の運用にかかる計算機リソースの利用費用を低く抑え，多数のプレイヤに対して常時サービス提供を行えること．

大量のデータをストアする手段としては，ペタバイト規模のデータを扱える BigTable [2]，Cassandra [3] などの分散データベースを用いることが考えられる．しかしこれらのデータベースは読み込み/書き込みのいずれかの性能を重視するように偏って設計されているため，ストアしている全格子データの読み書きが同時・高頻度に発生するような，本稿で述べるシミュレーション用途で発生するワークロードでは性能が低下する [4]．一方，大量の演算を高速に実行する手段として，ペタフロップス規模の演算性能を有するスーパコンピュータを用いることが考えられる．しかし，ゲームサーバとして 24 時間稼働し，ユーザに対して継続してサービスを提供する必要があることを考えると，スーパコンピュータの利用は運用コストが大きく，非現実的である．

本稿では上述した問題に対応し，クラウド上の安価でコモディティ化したコンピューティングリソースを用い，広大な空間を対象とするシミュレーションを実行しつつ，その空間に対して多数のクライアントがインタラクティブにアクセスできるサービスを実現するための分散システ

ムアーキテクチャを提案する．まず関連研究を述べた後に本アーキテクチャの構成と実装を示す．次に，インタラクティブな操作のための広域/詳細表示のシームレスな切替えの重要性について説明し，それを可能にするデータ分布を分散システム上に実現するデータ拡散レプリケーション手法を紹介する．最後に，実装した分散システムを用いた性能評価結果を示し，本手法によるインタラクティブシミュレーションの大規模化可能性を示す．

2. 関連研究

2.1 PGAS

Partitioned Global Address Space（PGAS）は，複数のプロセスが並列に動作する Simple Program Multiple Data（SPMD）型の並列処理において，大域的なメモリ空間を，分割されたメモリ空間とプロセスによって扱う計算モデルである．XcalableMP（XMP）[5] は PGAS に基づいた並列処理言語であり，XMP が定める指示文を記述して，データや処理の分割，各ノードへの割り当て，ノード間通信を行う．Unified Parallel C（UPC）[6] は同じく PGAS に基づいた並列処理言語であり，データ型に修飾子を付与することでノード上へのデータ分散を行い，UPC が提供する関数によってデータ通信を行う．いずれの言語も，典型的な並列処理を，性能を保ちながら高い生産性で記述できるという特徴がある．

本稿で述べるシミュレーション基盤は，メモリの参照局所性を有するシミュレーションプロセスを並列実行するという点で SPMD であり，一部のデータをノード間で共有する必要もあることから，その処理を PGAS に基づいて実装できる可能性がある．しかし，本シミュレータ基盤の特徴の 1 つであるデータ拡散は，ノードが局所的に近傍ノードとデータを交換し合うことで，自己組織的にノードネットワーク上にデータ分布を生み出す方式であり，PGAS の提供するパターン化されたデータ配置指示ではその処理記述が難しい．また，PGAS は主に High Performance Computing（HPC）分野において，Remote Direct Memory Access（RDMA）によって通信を行う高性能な PC クラスタ上で，演算を集中的に実行することのみを目的としている．これに対し，本稿のシミュレータ基盤は，クラウドコンピューティングの枠組みの中で，複数テナントのデータセンタにある，一般的な IP ネットワークで接続された安価なコンピュータ群上で，継続的なシミュレーション実行を行うことが目的であり，求められる性能と費用の要件が異なる．

2.2 GIS 用の分散データベース，分散処理

Geographic Information System（GIS）分野では空間情報を扱うための分散データベース，分散処理基盤が多く存在している．MySQL [7] などの Relational Database Management System（RDBMS）は空間データを効率的に検索するための空間インデックス機能を備えている．また，Redis [8] などの非リレーショナルデータベース（NoSQL）にも空間インデックス機能を持つものは多数存在する．DISTIL [9] はシステムを構成するノードのメモリ上にデータを配備したうえで，時間と空間を離散化した空間インデックスを構成し，そこに対する時空間の範囲問合せ処理応答を高速に行うインメモリの空間データ管理システムである．これらのシステムは大量の時空間データを効率良く保持・検索・変換するためのスケーラビリティを有している．しかし本基盤が想定するワークロードでは，数秒ごとのシミュレーションループのたびに，ストアしている全データの更新と読み込みが継続的に発生するため，このような状況下では，インデックスが有効に機能せず，処理性能が低下する可能性がある．また，いずれのシステムもベクターデータを対象として，円や点，多角形で表現されたオブジェクトどうしの距離比較や包含判定などの幾何学的な演算を行うことを想定している．これに対して本稿で想定しているシミュレータが対象とするデータは仮想世界を構成する格子上のラスタデータである．このため，ベクタデータを対象とした GIS 用の分散システムを本シミュレータの駆動に適用しても十分な性能を出すことができないと予想される．

一方, Guan は, 地理的なラスタデータを対象とした解析処理を汎用的に行うための枠組みとして, general-purpose parallel raster processing programming library（pRPL）[10] を提案している．gRPL は 4 分木ベースのインデクシング手法により対象のラスタデータを分割し，各領域を異なる計算機によって並列計算することでラスタ処理を高速に実行することを可能にしている．ここで，gRPL が想定しているラスタ処理は，たとえば農業・工業における土地の利用・被覆の経年変化を表すような cellular automata（CA）モデルである．本稿で述べるシミュレータも，仮想世界上で何かしらの現象を再現する CA モデルをはじめとしたラスタ処理を実行することを想定しており，本基盤と目的が類似している．しかし，gRPL はバッチ的に CA を実行することを想定しており，継続的かつインタラクティブにシミュレーションを実行し続けることが考慮されていない．インタラクティブなシミュレーションを可能とするためには，クライアントが広い空間領域を俯瞰的に遅延なく可視化できることが望ましい．しかし，gRPL では計算の対象とする空間の領域データが異なる計算機に分散して保持されるため，クライアントが広い範囲を可視化するためには，その領域範囲に含まれたデータを保持する複数の計算機からデータを集約する必要があり，その通信時間がインタラクティブ性を損ねる遅延原因となる．

2.3 MMORPG のためのシステム

MMORPG（Massively Multiplayer Online Role-Playing Game：大規模多人数同時参加型オンライン RPG）のためのアーキテクチャとして，プレイ空間を細かなエリアに分割して複数のサーバに割り当てたうえで，各エリアに対してアクセスしてくるクライアントへの応答処理を各サーバで分担することで負荷分散する方式が多く提案されている [11], [12], [13], [14], [15], [16]．文献 [12] では，サーバに割り当てる空間領域の量をサーバのシステムへの接続時間実績を考慮して変更することで空間データ保持の安定を図りつつ，1 サーバプロセスあたり 128 クライアントを CPU 負荷少なく導入できることを示している．空間の広さやクライアント数に対するスケーラビリティを確保するためにはこのような分割方式が効率的であるが，従来の方式では gRPL 同様に，プレイ空間を俯瞰的に広く可視化することが考慮されていない．

一般的に，大規模な空間データを可視化する際には，データ取得のための通信帯域消費や描画処理負荷を抑えるために，事前に作成した異なる詳細レベル（level-of-detail：LOD）を持つデータを用いることが有効である．多くの場合，空間データを格子（“タイル” とも呼ぶ）に分割したうえで，異なる LOD を保持するタイルから構成されるタイルピラミッドを生成してサーバに保存しておき，クライアントがそこにアクセスして可視化に必要な最小限のデータを取得することで描画を高速化させる手法が用いられる [17], [18]．MMORPG のための分散アーキテクチャでは各エリアを担当するサーバの管理や，プレイ空間全体にかかわる広域的な情報を保持するための，Central Server，World Observer，Super Node などと呼ばれる代表サーバの存在を仮定しており，この代表サーバに LOD の低い広域的なタイル情報を集約し，クライアントへ提供する方式が考えられる．しかし，代表サーバを介したデータ提供を行うと，クライアントからのデータ取得要求が代表サーバに集中しボトルネックとなる．

3. 提案方式

3.1 基本構成

本稿で提案するシミュレーション基盤は，格子（タイル）によって表現された仮想世界の空間データを格子状にパーティショニングした領域（タイルマップ）を，複数のノードが分担してメモリ上に保持する分散システムである．各ノードはクラウド上のコンピューティングインスタンス（仮想サーバ）上で動作させることを想定している．すべてのノードは役割に違いはなく，コンピューティングインスタンスを追加してノードを増やすことで仮想世界の広さを水平的にスケールする．**図 2** に本システムの基本構成を，**図 3** にノードの機能構成を示す．ノードは格子状に接続されており，次のタスクを実行する．

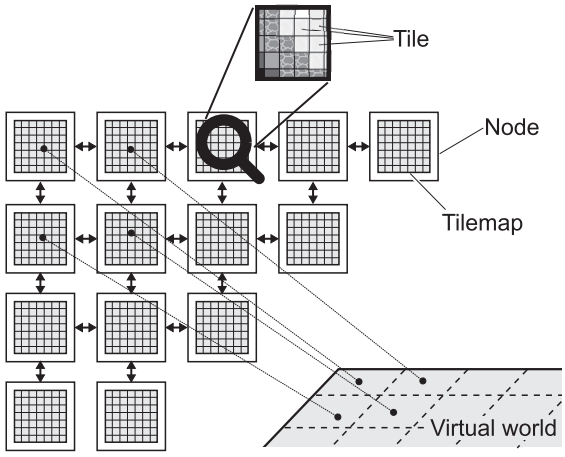

図 2 シミュレーション基盤の基本構成
Fig. 2 Overview of our simulation platform.

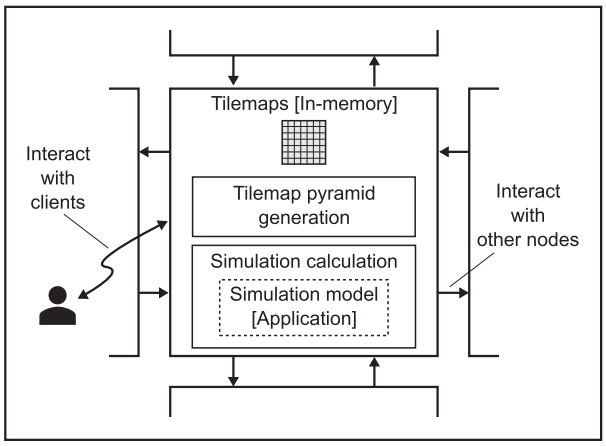

図 3 ノードの機能構成
Fig. 3 Functioning structure of a node.

タイルマップの時間発展計算

保持するタイルマップを特定の規則で更新し続けて空間の時間発展をシミュレートする．具体的には CA や偏微分方程式を離散化して得た差分方程式によって定められた近接タイルどうしの局所的な相互作用を，保持するタイルマップ内の全タイルに適用して更新するラスタ処理を数秒間隔で繰り返す．

タイルピラミッドの構築，更新

タイルマップの時間発展計算を実行するたびに，保持するタイルマップからタイルピラミッドを生成する．

タイルマップの送受信

近傍ノードから送信されてきたタイルマップを受信し，メモリ上に保持する．また，自ら生成するタイルマップを受信したタイルマップとあわせて近傍ノードへ送信する．

クライアントとのインタラクション

ゲームプレイヤが空間に対して何かしらの操作を行った際など，クライアントからの要求に応じてタイルマップの一部を更新する．また，クライアントに対

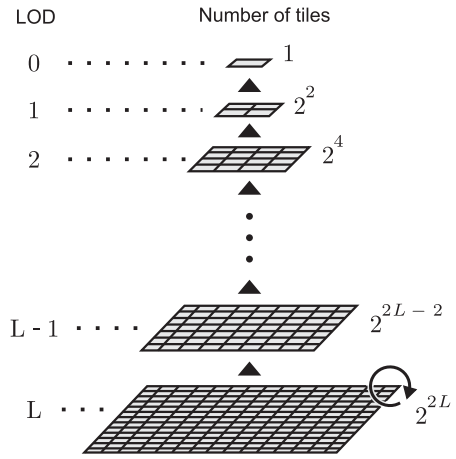

図 4 ノードが生成するタイルマップピラミッド
Fig. 4 Tile-map pyramid generated by a node.

して空間を可視化するためのデータを送信する．

3.1.1 ノードが生成・更新するデータ

ノードが自ら生成するタイルマップデータを**図 4** に示す．1 つのノードが保持するタイルマップの 1 辺の長さ（タイル数）を 2^L とし，これを LOD が L のタイルマップとすると，ノードはこのタイルマップに対して時間発展計算を行いタイルの値を数秒ごとに更新する．同時に，このタイルマップから，解像度を再帰的に減少させた L 個の異なる LOD $\{L-1, L-2, L-3, \ldots, 1, 0\}$ を保持するタイルマップを生成し，タイルピラミッドを構築する．

3.2 データの拡散レプリケーション

3.2.1 クライアントへのデータ提供

クライアントは仮想世界を可視化するためのデータを取得するため，ノードにアクセスする．クライアントからアクセスされたノードは，自らが保持するタイルマップデータをクライアントへ提供する．クライアントが仮想世界の広い領域を可視化するためには，アクセスするノード以外のノードが生成するタイルマップデータも必要となる．これらを得るためにクライアントが各個別のノードに対して個別に通信を行うと，それに要する通信時間がインタラクティブ性を損なう遅延の原因となる．そこで本システムでは，すべてのノードが，そのノードの周辺に存在するノードのタイルマップデータのレプリケーションを保持する．そして，クライアントからアクセスされたノードは，自ノードにリプリケートされたタイルマップデータを自らが生成したタイルマップデータとあわせてクライアントに提供する．こうすることで，クライアントが広い領域の可視化を行う場合であっても，複数のノードにアクセスする必要なく，通信量少なく可視化に必要なデータを入手できる．

3.2.2 空間可視化に適したデータ分布

Shneiderman は対話的可視化を前提とした情報可視化の設計指針として，まず広域表示で全体を把握し，次に興味のある領域を拡大し詳細を表示することを提唱している [19]．仮想世界をインタラクティブに可視化する際も，プレイヤはまず仮想世界を広域表示して全体を把握したうえで，興味のある領域を拡大して詳細表示をすることが予想される．多数のプレイヤがこの原則に従って仮想世界を可視化すると，必然的に広域表示のために必要なデータが，詳細表示に必要なデータと比較して頻繁にクライアントから求められることになる．したがって，ノードをつなぐネットワーク上には，解像度が粗く LOD の低い広域表示用のタイルマップはより多数のノードに重複して分布して読取負荷が分散されることが望ましく，一方で解像度が細かく LOD の高い詳細表示用のタイルマップは，限定されたノードに局所的に存在していればよい．**図 5** に，このような分布の例を模式的に示す．図 5 において，LOD が L のタイルマップは各ノードが 1 つずつ存在している．LOD が $L-1$ のタイルマップは，各ノードがマンハッタン距離で 1 以内の近傍ノードと相互に重複保持している．LOD が $L-2$ のタイルマップは，距離 2 以内の近傍ノードとタイルマップを相互に重複保持している．**図 6** に，プレイヤの視点に応じて必要となる LOD を模式的に示す．この図に示すとおり，プレイヤは広域表示をする場合であっても単一のノードにアクセスするだけで可視化に必要なデータを入手できる．

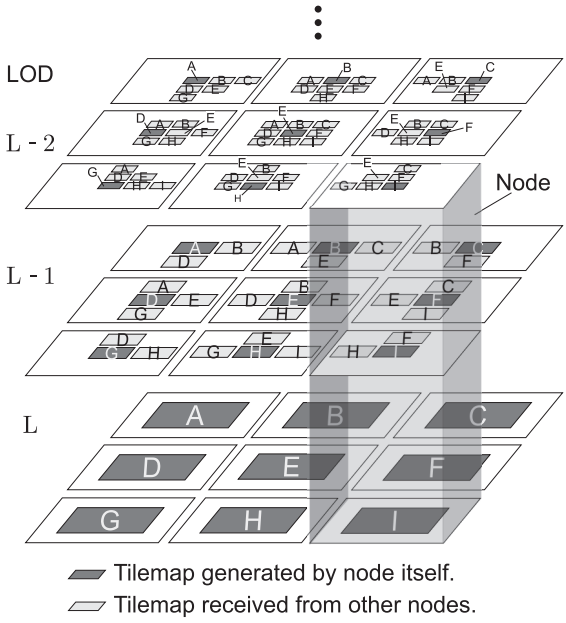

図 5 負荷分散のための理想的な空間データ分布の概念図
Fig. 5 Ideal spatial data distribution for load balancing.

3.2.3 データ伝播と拡散半径の設定

前述したデータ分布を生み出すため，各ノードは近傍（東西南北）の 4 ノードに対して自らが保持するタイルマップデータを配信する．データを受信したノードは，さらに近傍のノードに対して，受信したデータを自らのタイルマップデータとあわせて伝播させる．タイルマップデータには

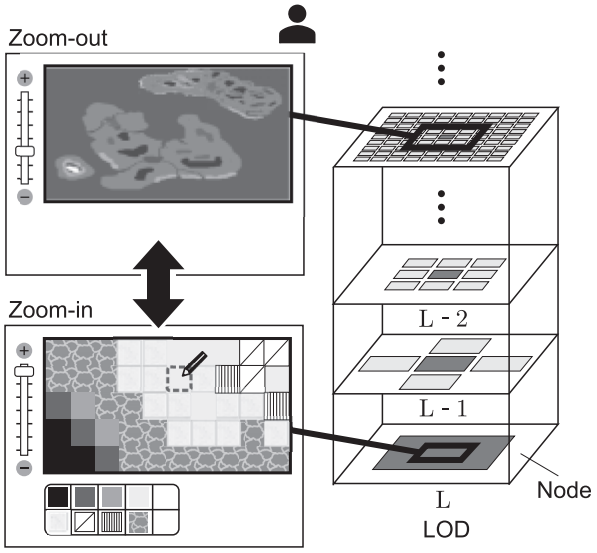

図 6 プレイヤの視点に応じたタイルマップへのアクセス
Fig. 6 Access to tile-maps according to player's viewpoint.

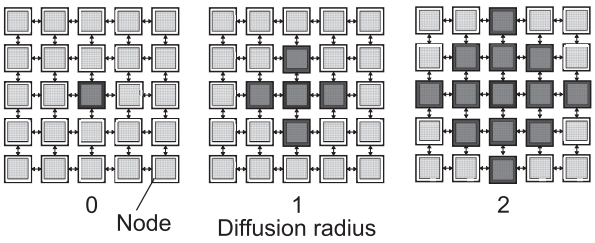

図 7 拡散半径に応じたデータの分布
Fig. 7 Data distributions according to diffusion radius.

拡散半径とホップ数が付与される．ホップ数はそのデータがノードを介して中継されるたびに加算され，ホップ数が拡散半径を上回った時点で，そのデータの伝播は打ち切られる．これをすべてのノードが繰り返すことで，あるノードから配信されたタイルマップデータは，時間経過とともにそのノードの周囲のノードへ拡散半径内を拡散していく．拡散半径に応じたノードネットワーク上のデータ分布の模式図を**図 7** に示す．ノードの基本的な動作をアルゴリズム 1 に示す．

拡散半径はタイルマップの LOD と LOD 切替タイル数 LOD switching tile count（LODstc）に基づき決定する．LODstc はクライアントが視野内で描画するタイルの最大数である．プレイヤが視点高度を上げズームアウトすると視野に含まれるタイル数が多くなるが，視野内のタイル数が LODstc に到達するたびに，LOD を 1 段階下げることで，描画タイル数を $\sqrt{LODstc}$ に軽減する．LODstc を 2^{2S} とおくと，LOD として k を持つタイルマップの拡散半径 r は，マンハッタン距離で

$$r = \begin{cases} 1 & (S < k) \\ 2^{S-k} & (S \geq k) \end{cases} \tag{1}$$

となる．

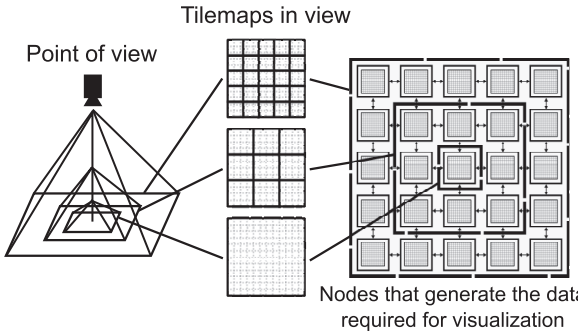

図 8 視点高度に応じて変化する必要なノードデータ
Fig. 8 Required nodes data that changes according to the viewpoint height level.

図 8 に，プレイヤの視点高度と，その高度において可視化に必要となるタイルマップを生成しているノードの範囲を示す．プレイヤの視点高度が十分低い場合，視野の範囲は局所的となるため，クライアントはアクセスしているノードが保持する LOD の高いタイルマップのみが必要となる．このため LOD の高いタイルマップは近隣のノードどうしが広く共有する必要がなく，拡散半径は小さな値となる．プレイヤが視点の高度を上昇させると，より広い範囲が視野に入り，アクセスしているノード以外のノードが保持するタイルマップも必要となるが，このときに用いる LOD は LODstc に応じて段階的に下がった低いものである．したがって，LOD が低いタイルマップはノード間で広く共有させるために大きな拡散半径が設定される．なお，LOD が 0 の状態で視点高度を上げ続けた場合，視野に含まれるタイルマップの数が LODstc に到達した時点で，それ以上の高度上昇を抑止する．つまり，プレイヤの視点の高度は LODstc によって上限が規定されることになる．また，ノードが保持するタイルマップの境界辺に含まれるタイルは仮想世界のパーティショニング境界に位置するため，これを更新するためには近傍のノードのタイルマップが必要となる．したがって式 (1) では拡散半径の最小値を 1 に定め，少なくとも直接接続している近傍ノードにはデータが拡散されるようにした．**表 1** に LOD に応じた拡散半径の値を，LODstc が 2^{12} である場合を例にとって示し，**図 9** にこのときのデータ分布状況を模式的に示す．なお，複数のノードに重複して存在するタイルマップに変更が生じてデータの一貫性がとれなくなることを防ぐため，クライアントが変更を加えることができるのは，そのノードが保持する LOD が最大である L のタイルマップのみとする．

3.3 同期と整合性

3.3.1 同期モードと非同期モード

本システムは，同期モードと非同期モードの 2 つの動作モードを有する．同期モードでは，ノードが世代 $t+1$ の時間発展計算を実行するとき，近傍すべてのノードから世

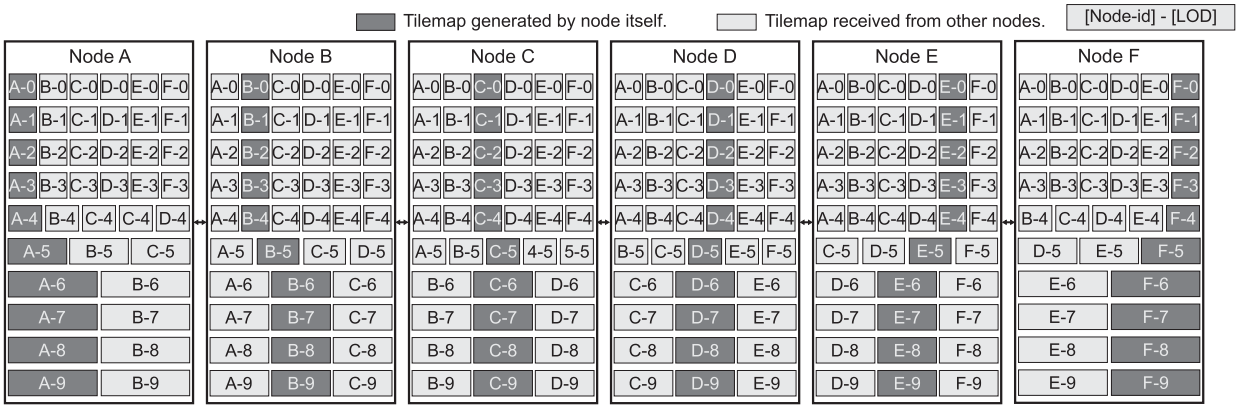

図 9 データ分布の例．説明のため，ノードが空間 1 次元で接続された単純な構成で示した
Fig. 9 An example of data distributions.

アルゴリズム 1 Main thread of node

```
Require: n ← list of neighbor nodes
  m ← INITIALIZE_OWN_TILEMAP
  while true do
    m ← UPDATE_OWN_TILEMAP(m)
    r ← RECEIVE_TILEMAP_FROM_NEIGHBORS
    p ← UPDATE_TILE_PYRAMID(m, r)
    for all tile ∈ p do
      d_r ← diffusion radius of tile
      d_h ← hop count of tile
      if d_r > d_h then
        d_h = d_h + 1
        set hop count of tile to d_h
        for all neighbor ∈ n do
          send tile to neighbor
        end for
      end if
    end for
    SLEEP(interval)
  end while
```

表 1 LOD と拡散半径の例
Table 1 An example of LOD and diffusion radius.

LOD	タイル数	拡散半径	LOD 切替高度
0	1	2^6	2^{14}
1	2^2	2^5	2^{13}
2	2^4	2^4	2^{12}
3	2^6	2^3	2^{11}
4	2^8	2^2	2^{10}
5	2^{10}	2	2^9
6	2^{12}	1	2^8
7	2^{14}	1	2^7
8	2^{16}	1	2^6
9	2^{18}	1	−

代 t のタイルマップデータを受信するまでバリア同期に入る．非同期モードで動作している場合には，近隣のノードからのデータ受信状況にかかわらず，時間発展計算を一定間隔で実行する．**図 10** に 2 つの動作モードの挙動を時間軸に沿って示す．同期モードでは，空間のパーティショニング境界に位置するタイルマップの境界辺のタイルを更新するために必要な近傍ノードのタイルマップの世代が自ノードと一致し，結果としてすべてのノードでシミュレーションの進行時間が揃う．また，どこかのノードで通信遅延などに起因するバリア同期遅れが生じた場合，その遅れが周辺のノードに連鎖波及し，システム全体のシミュレーション進行が遅延することになる．一方，非同期モードではシミュレーションの進行時間がノード間で完全に一致しないが，特定ノードの遅延に起因したシステム全体の遅延発生は生じない．同期モードと非同期モードではシミュレーションの結果が異なるが，ゲーム用途では非同期モードでも問題は生じない場合が多い．

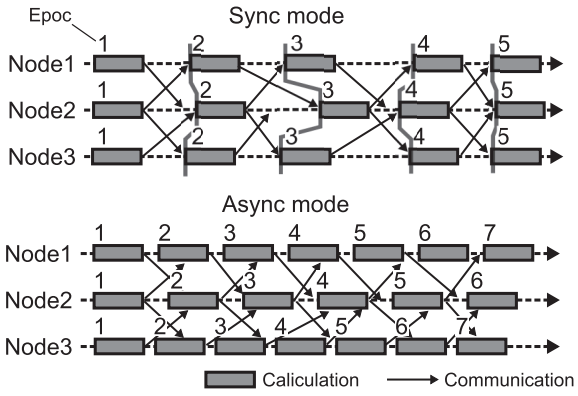

図 10 同期モードと非同期モード
Fig. 10 Sync mode and async mode.

3.3.2 タイルマップの整合性

同期・非同期モードにかかわらず，ノードが保持するタイルマップには，そのデータが拡散過程でたどった経路長に応じたレプリケーションラグが存在する．具体的には，あるノードが保持するデータが経路長 p をたどって拡散したものだとすると，時刻 t の時点では，そのデータの世代は $t-p$ となる．このため，プレイヤが仮想世界を可視化した際には，LOD が低い広域な表示ほど古い世代の空間が表示されることになる．しかし，通常のシミュレーショ

ンでは空間を構成するタイルの多くは定常状態で変化しないことが予想され，空間の局所的な変化が積み重なって広域的に視認できる程度の広い範囲に変化が生じるためにはそもそも時間がかかるため，レプリケーションに遅延が生じても実用上は大きな問題とならない．

3.4 クライアントがノードを探す方法

クライアントが目的の場所のデータを持つノードの宛先（IP アドレス）を得るためには，大きく分けて次の 2 通りの方法が考えられる．

集中保持方式

ノードの数が限定されており，かつ，それらのアドレスに変化が生じない場合は，クライアントが，目的の場所の座標から，その位置のタイルデータを保持するノードのアドレスを引くためのリストを保持しておけばよい．ノードの数が多いか，あるいはアドレスが頻繁に更新されるような場合は，座標を元にアドレス情報を提供するネームサーバを用意することが考えられる．ただしこの場合はネームサーバがクライアント数やノード数増加のボトルネックになりがちである．

分散保持方式

ノードが，周囲に拡散するタイル情報に対して自らのアドレスを付与しておき，可視化する領域を移動・拡縮する際には，タイル情報に付与されたアドレスをたどってアクセスする方式である．クライアントは初回にアクセスするノードのアドレスをいくつか保持しておき，そこからランダムに選択したノードから取得するタイル情報に含まれるアドレスをたどってアクセス先を切り替える．ただし，LODstc によって規定される視点の上限高度が低い場合は，俯瞰表示して得られるタイルに目的の領域が含まれない可能性があるため，この場合は目的の領域が可視化範囲に含まれるまでユーザが画面スクロールする操作が必要となる．

4. 実装

提案した手法に基づくシステムを実装した．ノードはタイルマップデータを zip 圧縮したうえで Socket（TCP）で近傍ノードと送受信しあう．ノードは Web サーバとしても振る舞い，アクセスしてきたクライアントに Web コンテンツを提供する．Web コンテンツには HTTP もしくは WebSocket を介してノードと双方向に通信することでインタラクティブに仮想世界を可視化・操作するプログラムが含まれている．プレイヤはアプリケーションをインストールすることなくブラウザを用いてノードにアクセスし，シミュレータを利用できる．ノードは 1 つのプロセスとして常駐動作し，シミュレーション実行部，他ノードとのデータ送受信部，クライアントとのインタラクション部がそれぞれ独立のスレッドで動作する．実装言語は Java であり，タイルマップデータは ByteBuffer クラスによって非ダイレクトバッファとして保持した．また Web コンテンツに含まれるプログラムは JavaScript で実装した．

5. 評価

5.1 評価観点

提案方式の有効性を確認するため，実装したシステムを次の項目で評価した．

（評価 1）空間の広さに対するスケーラビリティ

合計ノード数の増加にともなう個々のノードのリソース消費変化傾向を定量的に確認し，仮想世界を構成するシミュレーション空間の広さをスケールアウト可能なことを示す．

（評価 2）リクエスト数に対するスケーラビリティ

クライアントからのリクエスト数の増加にともなう負荷の増加傾向を，代表サーバ方式と比較する．まず理論的な解析を行ったうえで，実験により定量的な差を示す．

5.2 評価環境・条件

評価にあたり，ノードは AWS EC2[*2]のコンピューティングインスタンスの上で動作させて計測した．1 つのインスタンスにつき 1 つのノードプロセスを実行する構成をとり，複数のインスタンスを同一リージョン，同一アベイラビリティゾーンの中に配備して分散システムを構成した．インスタンスの OS には CentOS 7 を用いた．**図 11** に評価環境におけるシステム構成を示す．

評価時に動作させるシミュレーションとして，次の遷移則によって定めた単純な CA モデルを用いた．

- セルは状態 $\{0, 1, 2, 3, 4, 5\}$ のうち，いずれかの状態をとり，初期値はここからランダムに割り当てる．
- 更新対象のセルの東西南北に近接する 4 つのセルを，更新対象セルの近傍セルとして定義する（ノイマン近傍）．

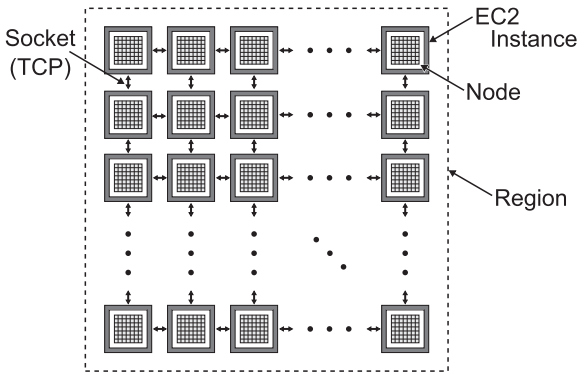

図 11 評価環境におけるシステム構成
Fig. 11 Evaluation environment.

[*2] Amazon Web Services, Elastic Compute Cloud

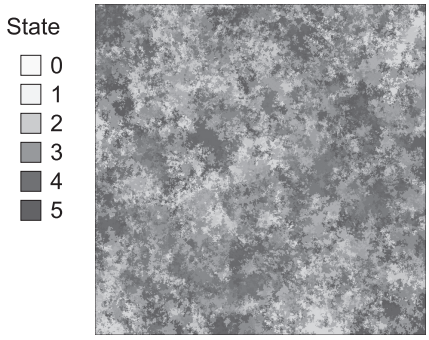

図 12 CA モデルによって駆動されるシミュレーション空間
Fig. 12 A virtual world driven by the CA model.

- 近傍セルの状態がすべて同一であれば，更新対象セルの状態もその状態に変化させる．
- 近傍セルの状態がすべて同一でなければ，更新対象セルの状態を，自身と近傍セルのなかからランダムに選択したセルと同じ状態に変化させる．

このモデルは Procedural Content Generation（PCG）アルゴリズムの一種である．単純な規則でありながら，複数のノードをまたぐ広域的な構造が創発的に生成され，複数のノードが問題なく協調動作しているか確認するうえで都合が良いために今回の評価に用いた．**図 12** にこのモデルを動かしている仮想世界をクライアント側で可視化した様子を示す．

5.3 評価 1

5.3.1 評価方法

空間の広さを拡大したときの性能を検証するため，ノードの数を 2^2 から 10^2 まで増やし，それぞれの場合において，ノードプロセスを実行しているインスタンスの CPU 消費率，メモリ使用量，ネットワーク帯域消費量を計測した．ノードが保持するタイルマップのサイズの 1 辺の長さは 2^9 とした．したがってタイルピラミッドには 10 個の異なる LOD を持つタイルマップが含まれることになる．LODstc は 2^{10}，タイルマップを構成するタイルのデータサイズは 2 バイトとした．また，動作モードは非同期とし，タイルの更新，および近傍ノードへのデータの配布間隔を 1 秒とした．ノードを動かすコンピューティングインスタンスは t2.micro（vCPU：1，メモリ：1 GiB）を用いた．

5.3.2 結果と考察

評価 1 の結果を**図 13** に示す．図 13 は仮想世界を構成するノードの全体数変化に対する，1 つのノードのリソース消費変化を示している．各ノードは毎秒消費リソースの計測を行い，10 分間の測定を実施した．グラフで示した値はノードが個別に求めた 10 分間の計測平均値の全ノード分の平均値である．なお，メモリ消費量は Java のヒープメモリ領域内でのメモリ利用率から算出した．ヒープメモリの全体サイズは 950 MB に設定した．図 13 では，ノード数が 4^2 までは，ノード 1 台あたりの CPU 利用率とネットワーク消費帯域がほぼリニアに増加している．これは仮想世界の広さの増大にともなって，各ノードに対して，そのノードを中心したより広い範囲のノードのタイルマップデータが拡散されてくることが原因である．一方でノード数が 8^2 を超えるとすべてのリソース消費量に大きな増加がなくなる．これは仮想世界がさらに広くなるにつれて，個々のノードに対して，そのノードの遠方に存在するノードのデータが拡散されてこなくなるためである．つまり本システムは，仮想空間の全体ノード数を増やしていっても，その数が一定量を超えた時点で，全体ノード数増加にともなうノード 1 台あたりの負荷増加は漸減し，いずれかの時点，それ以上ノードを増やしても，個々のノードに対する負荷が変化しなくなる．本評価では計測を通じてその挙動を再現でき，ボトルネックなしに仮想世界の広さをスケールアウトできることを確認した．

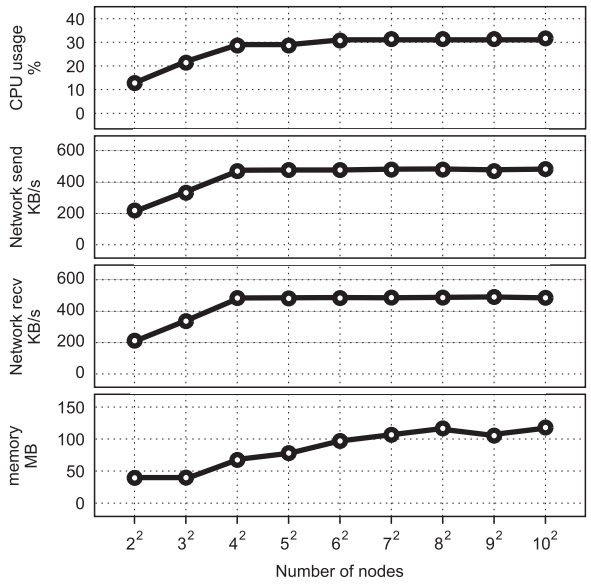

図 13 ノード数増加にともなう負荷変動
Fig. 13 Load fluctuations according to the increase in the number of nodes.

なお，仮想世界を広くとるために，単一ノードの性能を高めて，保持するタイルマップのサイズを増やすことも考えられる．たとえばエリアあたり AWS のインスタンス m5.large を 16 台で担当している構成を，m5.4xlarge を 4 台，あるいは m5.8xlarge を 1 台に置き換えても，いずれも vCPU 数の合計は 32 個，メモリ量の合計は 128 GiB となり，時間あたりの稼働費用も変わらない[*3]．しかし，空間の広さを運用開始後も柔軟に少しずつ広げていくためには，比較的安価で性能の低いコンピューティングインスタンスを漸増的に投入していける方が望ましく，また，クライアントからのリクエストによるノードあたりのネットワーク帯域負荷も，インスタンス数が多いほうが分散される．し

[*3] 2020 年 8 月現在．

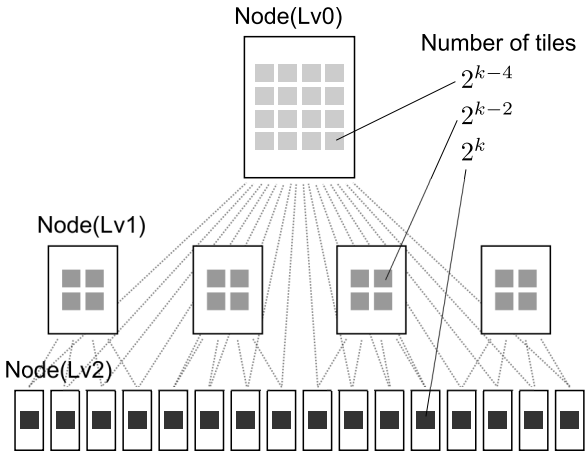

図 14 代表サーバ方式の簡易モデル
Fig. 14 Simple model of central server method.

かしながら，空間面積あたりのノード数を増やして空間を細分化しすぎると，ホップ数の多いより遠方のノードへもタイルデータを拡散させる必要が出てくるためノード間通信量が増大し，これを防ぐために LODstc を低くとって拡散半径を狭めると，プレイヤが俯瞰できる領域が狭くなる．また，ノード性能のボトルネックはタイルマップデータを保持するためのメモリ量にあるため，基本的にはメモリ搭載量を基準にインスタンス性能を決めればよい．しかしシミュレーションの内容によっては格子更新のための演算負荷が大きくなるため，この場合は CPU のコア数を基準にインスタンスを選定する必要がある．

5.4 評価 2

5.4.1 代表サーバ方式との理論比較

クライアントからのアクセス量に対する負荷の変動について，**図 14** で示した簡易的な代表サーバ方式モデルを対象として，提案方式と比較した．ここではノードの負荷は単位時間あたりにノードが送受信するタイル数に比例すると仮定してその比較を行った．図 14 は，代表ノード Lv0，Lv1 が Lv2 のノードデータを保持する構成を示している．Lv2 ノードはシミュレーションを実行しつつ単位時間あたりに 2^k のタイルを更新し，Lv1 ノードに対して LOD を 1 つ下げた 2^{k-2} 個のタイルを，また，Lv0 ノードに対しては LOD を 2 つ下げた 2^{k-4} 個のタイルを送信する．同レベルのノードに対してタイルの送受信は行わない．このシステムに対してクライアントから，空間上の場所と表示縮尺（LOD）をランダムに指定したうえで，その情報を持つノードに対して直接リクエストを送信する場合，単位時間あたりに x 回のリクエストが発生したときに，各レベルのノードが送受信するタイル数はリクエスト数に確率的な係数を掛けて，おおむね式 (2) のとおりとなる．

$$\mathrm{Lv0}: \underbrace{16\times 2^{k-4}}_{recv}+\underbrace{\frac{1}{3}\times 16\times 2^{k-4}\times x}_{send}$$

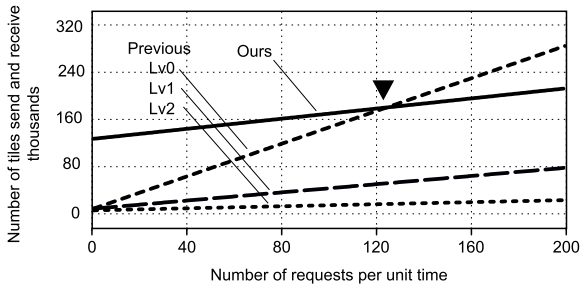

図 15 リクエスト数増加に対するタイル送受信数変化傾向の予測
Fig. 15 Prediction of tile send and receive fluctuations according to the increase in the number of requests.

$$\mathrm{Lv1}: \underbrace{4\times 2^{k-2}}_{recv}+\underbrace{\frac{1}{3}\times\frac{1}{4}\times 4\times 2^{k-2}\times x}_{send} \tag{2}$$

$$\mathrm{Lv2}: \underbrace{2^{k-2}+2^{k-4}+\frac{1}{3}\times\frac{1}{16}\times 2^{k}\times x}_{send}$$

これに対して，提案方式によって 4×4 の格子状に結合した各ノードの単位時間のタイル送受信数は，

$$\alpha = 16\times 2^{k-4}+11.75\times 2^{k-2}+2^{k}$$

とおいて，

$$\underbrace{3\times\alpha}_{recv}+\underbrace{3\times\alpha+\frac{1}{3}\times\frac{1}{16}\times\alpha\times x}_{send} \tag{3}$$

となる．式 (3) は，比較対象と条件を揃えるため，ノードが保持するタイルの LOD は 3 段階とし，それぞれのタイルマップのタイル数を 2^k，2^{k-2}，2^{k-4}，拡散半径は LOD が高い順から 0，3，6 とした．また，ノード群の端や角に位置するノードは結合する近隣ノードが少なく送受信タイル数も若干少なくなることを考慮した平均値を適宜係数に用いている．これらの予測式を用いて，単位時間あたりのリクエスト数の増加に対する，各ノードの送受信タイル数変化を予測したグラフを**図 15** に示す．図 15 は，リクエスト数が少ない場合は提案方式のノードのタイル送受信数がデータの拡散を行う分だけ多くなり，従来の代表ノード方式よりも負荷が高いが，リクエスト数の増加に応じて従来方式の Lv0 ノードのタイル送受信数が提案方式を上回り，システム全体として判断すると提案方式のほうが負荷の偏りが少なくなることを示している．

なおグラフは式に $k=12$ を代入して作成したが，定性的な傾向は k の値によらず同じである．また，この予測式はリクエストの表示縮尺がランダムと想定したものであり，広域表示よりも詳細表示のリクエスト割合が多い場合には，代表ノードにかかる負荷が抑えられ，提案方式よりも代表ノード方式の方が有利になる．逆に広域表示の割合が多い場合は，提案方式の方がより有利になる．

5.4.2 シミュレーションによる確認

図 14 で示した代表ノード方式による従来方式を $k=12$

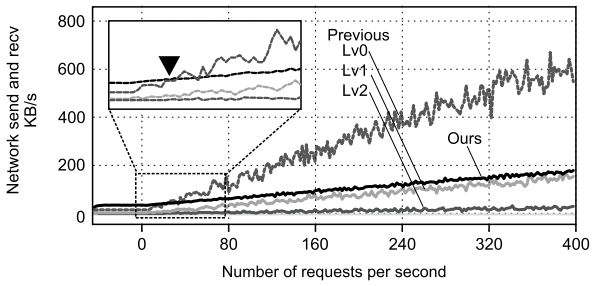

図 16 リクエスト数増加に対するネットワーク送受信量変化
Fig. 16 Network traffic fluctuations according to the increase in the number of requests.

としたうえで，前節で述べた条件と同じリクエストを秒間0〜400回まで増加させながら稼働させ，各ノードのネットワークの送受信状況を計測した．また，提案方式にも同様の負荷をかけ，前項で述べた負荷傾向予測の妥当性を検証した．なお，本評価ではノードに対して負荷が集中する可能性を考慮しt2.microよりも若干性能の高いt2.largeインスタンスを用いた．

5.4.3 結果と考察

図 16 に，シミュレーション結果を示す．図 15 では縦軸がノードあたりのタイル送受信数，横軸が単位時間あたりのリクエスト数であるのに対応し，ここでは縦軸がノードあたりのネットワーク送受信量平均，横軸を 1 秒あたりのリクエスト数とおいた．代表サーバ方式の Lv1 ノードは 4 台分，Lv2 ノードは 16 台分，また提案方式のノードは 16 台分の平均値の推移を示している．この図は，各ノードの負荷増加傾向がおおむね前節で述べた予測どおりであることを示しており，特に代表サーバ方式の Lv0 ノードは，予想どおりリクエスト数の増加にともなって，提案方式のノードと比較して負荷が高まることが確認できた．なお，図 16 内に三角形で示した，Lv0 ノードの負荷が提案方式ノードの負荷を上回るときの秒間リクエスト量は，図 15 内で示した予想量よりも少なかった．これは Lv0 ノードは通信回数が多くなるため，その度に発生する通信ヘッダ分のデータによってデータ送受信量が理論値よりも多くなったこと，およびノード間通信はデータを zip 圧縮して行うため，ノード間通信を多く行う提案方式は実際のデータ送受信量が理論値よりも少なくなったことが理由と考えている．

6. おわりに

6.1 提案と実装，評価のまとめ

本稿では，大規模なインタラクティブシミュレーションを可能にするための課題を述べたうえで，それらを解決するための基盤技術を提案・評価した．提案方式はクラウド上の安価な計算ノードを格子状に接続する分散システムであり，そこに空間データを拡散することで仮想世界の可視化に適した分布を生み出すことを特徴とする．評価の結果，ノードを追加することでシミュレーション空間の広さをリニアにスケールアウトできることを確認した．また，多数のクライアントからリクエストが発生している状況下で，従来方式よりも負荷の偏りが少なくなることを確認した．

また，提案方式はシステムの構成要素であるノードはすべて均質で同じ振舞いをし，ノードによる機能的，役割的な差異がない．さらに，負荷分散のためのデータレプリケーションを中央集権的な制御に基づいて行うのではなく，各ノードが自律的に近傍ノードと相互作用を重ねることで，空間可視化に最適なデータのレプリケーションが創発的に実現される．これらの均質性と自律性という特徴から，ノード群の構成が単純となり，単一障害点も存在せず，システム全体の保守性と可用性を高めることができると考えている．一方で，提案したシステムは，ノードが保持するタイルマップの世代が，LOD によって異なる，また非同期モードではノードごとのシミュレーション進行速度が異なるなど，データに対する整合性や一貫性を完全には保障していないことに注意されたい．

6.2 今後の課題

提案，実装したシステムに残された課題を次に述べる．

ノード故障時の対応

ノードに障害が発生して機能停止した場合でも，他のノードが障害ノードの代替を行うなどしてサービスを停止せずに継続する仕組みを検討する必要がある．

アクセス負荷の空間的偏りへの対応

実際の運用では，仮想世界には多くのプレイヤが集まって混雑する領域と，プレイヤが 1 人も存在しない領域が存在することが考えられる．このため，負荷に応じてノード自ら拡散半径を一時的に大きく設定し，周囲へ情報をレプリケーションする範囲を広げつつ，周囲のノードへクライアントに対する対応分担を依頼するといった負荷分散機構が必要となる．

6.3 他のアプリケーション例

本シミュレーション基盤を使ったゲーム以外の応用として，たとえば交通シミュレーションが考えられる．道路網を表現したデータをノードが分担して保持し，その道路上を走行する車両の挙動を CA などで表現したうえで実行すれば，都市レベルで大規模に交通流をシミュレーションできる．また，実際に都市を走行する車両が定期的にサーバに対してアップロードする CAN（Controller Area Network）データの入力をシミュレーションに対する介在と見なし，これによって仮想世界上を走行する車両の位置や速度を補正しながらシミュレーションを継続すれば，より現実世界の状況に即した交通状況の把握と予測が可能となる．

6.4 結論

本稿では，階層的にLODを持つ分散データ構造を持ち，可視化のためのデータをLODに応じた半径内に拡散するアーキテクチャにより，広大な空間を対象としたインタラクティブシミュレーションを低コストで実現できる計算基盤を提案した．実運用のためには前節で述べた残課題が存在するが，大規模化のための最も大きな課題である大量の空間データ保持と，大量の演算実行を両立できる見込みを，AWS EC2を使い，空間スケーラビリティ計測によって示した．また，一般的な代表サーバ方式と比較した本手法のメリットを理論解析と実験によって示した．

参考文献

[1] Nebel, S., Schneider, S. and Rey, G.D.: Mining Learning and Crafting Scientific Experiments: A Literature Review on the Use of Minecraft in Education and Research, *Journal of Educational Technology & Society*, Vol.19, No.2, pp.355–366 (2016).

[2] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data, *ACM Trans. Comput. Syst.*, Vol.26, No.2 (2008).

[3] Lakshman, A. and Malik, P.: Cassandra: A Decentralized Structured Storage System, *SIGOPS Oper. Syst. Rev.*, Vol.44, No.2, pp.35–40 (2010).

[4] Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R. and Sears, R.: Benchmarking Cloud Serving Systems with YCSB (2010).

[5] XcalableMP Specification Working Group: XcalableMP Language Specification Version 1.4 (2018), available from ⟨https://xcalablemp.org/download/spec/xmp-spec-1.4.pdf⟩.

[6] UPC Consortium: UPC Language Specifications Version 1.3 (Nov. 2013), available from ⟨https://upc.lbl.gov/docs/user/upc-lang-spec-1.3.pdf⟩.

[7] Oracle Corporation and/or its affiliates: MySQL, available from ⟨https://www.mysql.com/⟩.

[8] Redis, available from ⟨https://redis.io/⟩.

[9] Patrou, M., Alam, M., Memarzia, P., Ray, S., Bhavsar, V., Kent, K. and Dueck, G.: DISTIL: A distributed in-memory data processing system for location-based services, pp.496–499 (2018).

[10] Guan, Q.: pRPL: An open-source general-purpose parallel Raster Processing programming Library, *SIGSPATIAL Special*, Vol.1, pp.57–62 (2009).

[11] Jardine, J. and Zappala, D.: A hybrid architecture for massively multiplayer online games (2008).

[12] 金田裕剛，高橋ひとみ，斉藤匡人，間 博人，徳田英幸：P2P型大規模マルチプレイヤオンラインゲームにおける領域負荷分散性の一考察 (2007).

[13] Vleeschauwer, B., Van Den Bossche, B., Verdickt, T., De Turck, F., Dhoedt, B. and Demeester, P.: Dynamic microcell assignment for massively multiplayer online gaming, *4th ACM SIGCOMM Workshop on Network and System Support for Games*, pp.1–7 (2005).

[14] 豊国浩平，西村浩二，相原玲二：複数サーバを用いた大規模仮想空間の分散管理，技術報告 98 (1999-DSM-016).

[15] Iimura, T., Hazeyama, H. and Kadobayashi, Y.: Zoned federation of game servers: A peer-to-peer approach to scalable multi-player online games, pp.116–120 (2004).

[16] Lee, H.-H. and Sun, C.-H.: Load-Balancing for Peer-to-Peer Networked Virtual Environment, *Proc. 5th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames '06*, p.14–es (2006).

[17] Guo, N., Xiong, W., WU, Q. and Jing, N.: An Efficient Tile-Pyramids Building Method for Fast Visualization of Massive Geospatial Raster Datasets, *Advances in Electrical and Computer Engineering*, Vol.16, pp.3–8 (2016).

[18] Wang, L., Chen, D., Deng, Z. and Huang, F.: Large scale distributed visualization on computational Grids: A review, *Computers & Electrical Engineering*, Vol.37, pp.403–416 (2011).

[19] Shneiderman, B.: The eyes have it: A task by data type taxonomy for information visualizations, *Proc. 1996 IEEE Symposium on Visual Languages*, pp.336–343 (1996).

山岡 久俊 （正会員）

2009年早稲田大学大学院情報生産システム研究科修了．同年株式会社富士通研究所入社．同社にて分散ストリーム処理基盤の研究開発に従事しつつ，2019年より筑波大学大学院システム情報工学研究科に所属．

蔡 東生 （正会員）

1985年スタンフォード大学電気工学科研究助手．1989年東京大学大学院工学系研究科修了．工学博士，Ph.D.．1989年神戸大学工学部助手．現在，筑波大学大学院システム情報工学研究科准教授．科学技術計算と可視化等の研究に従事．