空間の広域可視化を可能とする
大規模分散システムの開発

2022年3月

山岡　久俊

空間の広域可視化を可能とする
大規模分散システムの開発

山岡　久俊

システム情報工学研究科

筑波大学

2022年3月

概要

近年，多人数が参加可能で，利用者同士が仮想空間上で自由に行動をし，コミュニケーションをとるゲームやツールの利用が浸透しつつある．また，IoT（Internet of Things）によって，現実世界から取得できるセンサデータの種類や量が増えており，これらを仮想空間上で重畳表示することで，現実世界の状況を適切に把握するシステムへのニーズが高まっている．いずれも，扱える空間を広くするために大規模な空間データを扱えること，および利用者の操作に応じてその空間を遅延なくインタラクティブに可視化できること，また，センサデータを入力として受け付ける場合はアップロードされてくる大量・多様なセンサデータを柔軟に変換・加工できるようにすることが，そのサービスやシステムの魅力や価値向上のために重要となる．本研究では，これらを可能にするための，空間データの生成・保持・更新・可視化やセンサデータの変換・加工を大規模かつ効率的に行う研究に取り組む．

まず 1 つめは，大規模な空間データを保持・更新しつつ，クライアントからの要求に応じて表示の縮尺を柔軟に変えながら空間を可視化できる基盤を提案する．ここでは，複数のノードに分散して存在するデータを集約する際にかかる時間と負荷を解消するため，ネットワークを介して相互接続する複数のノード間でデータの交換を継続的に行い，可視化のために理想的なデータ分布を生み出す方式を提案し，評価実験によってその効果を検証する．

２つめは，仮想空間上で利用者が違和感を感じない自然なインタラションを実現するため，利用者が操作するキャラクタなどの特定の地点を中心とした近傍領域を生成し，その領域内に非言語情報が伝達する範囲を限定するなどの制御を行うための近傍領域モデルとその高速な計算方法を提案する．ここでは，空間上の構造物や障害物の配置関係に応じて柔軟に形状を変える近傍領域をポテンシャル関数によってモデル化し，偏微分方程式の近似解法である代用電荷方を応用することで，ポテンシャル分布を高速に計算する方法を提示し，評価実験によって提案手法の効果を検証する．

３つめは，継続して発生する大量の IoT デバイスから取得できるセンサデータを 1 つめで提示した基盤に入力するための前処理を行うストリームデータ処理基盤を提案する．ストリームデータ処理を行う上での課題を挙げ，これらに対応するために，並列分散ストリームデータ処理基盤の上で，低レベルなストリームデータ処理演算によって構成した処理フロー上に，オブジェクトとプラグインという概念を組み合わせて処理を記述する抽象レイヤを設けたストリームデータ処理基盤を提唱し，サービスを試作した上で提案方式の効果について考察する．

目 次

表 目 次

図 目 次

第1章　序論

1.1　研究の背景と関連技術の動向

近年，多人数が参加可能で，利用者同士が仮想空間上で自由に行動をし，コミュニケーションをとることができるゲームやツールの利用が浸透しつつある [130]．また，IoT（Internet of Things）によって，現実世界から取得できるセンサデータの種類や量が増えており，これらを仮想空間上で重畳表示することで，現実世界の状況を適切に把握することへのニーズが高まっている [26]．図 1.1 に，このようなゲームやツールの実現のために必要となる技術を俯瞰的に示す．

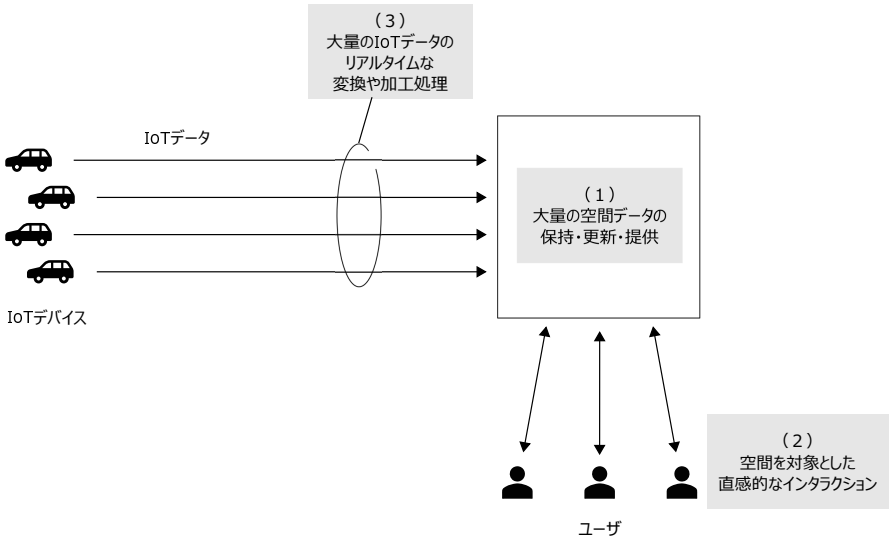


図 1.1: 大規模な空間データの可視化に必要となるシステム・技術

図 1.1 で示した要素のうち（1）「空間データの保持・更新・提供」は，広大な仮想空間を表現する大量の空間データを保持し，また，その一部や全てを更新し，それを複数の利用者に対して提供することを目的とした基盤技術である．（2）「空間を対象とした直感的なインタラクション」は，（1）によって提供される仮想空間を対象として，空間をインタラクティブに探索したり，空間上で他の利用者とコミュニケーションをとるために効率的なインタラクションを行うための技術である．（3）「IoT データのリアルタイムな変換や加

工処理」は,（1）によって保持される仮想空間を，実世界の状況に連動して更新するために，大量の IoT デバイスからアップロードされるセンサデータをリアルタイムに変換・加工する技術である．これら 3 つの技術領域に関する動向を以降にそれぞれ述べる．

1.1.1 空間データの保持・更新・提供を行うシステムの動向

空間データを扱うための代表的な枠組みに GIS（Geographics Information System）がある．GIS は，地理空間データを保持した上で，そこに地理情報を重ね合わせたり，例えばある地点から指定した距離内に存在する建物や構造物を抽出して視覚的に把握するバッファ分析などの空間解析機能を有している．GIS は空間データの保存や検索に適した空間索引，演算機構を備えた空間データベースを中心に構成される．空間データベースは，RDBMS（Relational Database Management System）をベースとするもの [18, 23] が一般的であるが，増加する空間データの量や種類に対応するために，データの読み書きの速度が RDBMS よりも優れている NoSQL をベースとするもの [12] も存在している．近年はネットワークを介して多くの利用者に空間情報を提供する WebGIS や LBS（Location Based Service）の台頭に伴い，大規模なクラスタによって NoSQL データベースを構成し，地球規模の広さの空間情報を大量の利用者に提供することが可能 [52] となっている．

空間情報を広域に可視化するためには，広い範囲を対象として，その範囲内に存在する空間データをデータベースのクラスタから収集する集約処理が必要となる．大規模クラスタによって構成する空間データベースはペタバイト級のサイズの空間データ保持を可能にする一方で，空間データの集約処理を行うと，データの規模によってはその処理に数百〜数千秒の時間を要する [30, 63]．既存の空間データベースにおける空間データの集約処理は，空間データのバッチ的な解析や，不定期な問合せに応答する際に行うことを目的としており，処理にこのような時間を要しても実用に足る場合が多い．しかし，常に変化し続ける空間データを広域かつ継続的に可視化することを考えると，頻繁かつ継続的な集約処理の実行は現実的ではない．そのため，空間データの保持・更新・広域可視化を目的として，空間データを高頻度かつ継続的に集約する方式を検討する必要がある．

1.1.2 空間を対象とした直感的なインタラクションに関する技術の動向

リモートワークの普及に伴い，利用者が仮想空間上に存在する利用者自身の「分身」を表すキャラクターであるアバターを操作し，アバターを介して利用者同士がコミュニケーションをとるツールやゲームに対する需要が増加している．また，仮想現実（Virtual Reality）

や拡張現実（Augmente Reality）技術を駆使して，仮想空間上で利用者同士がより高度なコミュニケーションを行うメタバースの概念も提唱されており，より臨場感や没入感を感じられるコミュニケーションを可能とする技術に対する需要が高まっている [123].

oVice™[1]や Spatial Chat™[2]は利用者同士が仮想空間上で交流するオンラインネットワーキングツールとして利用者を増やしているサービスである．いずれも利用者が操作するアバター同士の距離や向いている方向によって音声を始めとした情報の伝達範囲や強弱を制限し，これによって仮想空間上で「立ち話」をしているような臨場感を演出している点が特徴である．今後のコミュニケーションツールやゲームにおいても，利用者の操作するアバター同士の位置関係に応じた情報提示，あるいは物理的な事象が伝わる範囲の制御を行うことで，実空間にいる時に似せた状況を模擬的に再現することが重要となる．このような制御は，物理シュミレーションによって音や光の到達範囲を求めることで実空間と同様の状況を近似することができる [93,96]．しかし，一般的に物理シミュレーションを実行するには高次元の連立方程式を解いたり，大量のループ計算処理を行う必要があり多くの計算時間がかかる．そのため，多数の利用者を対象としてサービス提供することを考えると，この計算をサーバで行うことは非現実的である．また，利用者側（クライアント側）でこの計算を個別に実行する場合，その端末はスマートフォン等の携帯端末である可能性が高く，バッテリ消費が増加する物理シミュレーションを実行することは望ましくない．このため，空間上で物理的な事象が伝わる範囲を実世界と同様になるように近似しつつも，処理負荷の低い軽量な計算手法が必要となる．

1.1.3 IoT データの変換・加工処理に関するシステムや技術の動向

IoT デバイスの急速な普及に伴い，IoT デバイスからアップロードされるセンサデータの種類・量が増加している [27]．これに伴い，アップロードされたデータを一旦データベースに蓄積した後で遡って分析するのではなく，データをデータベースに蓄積せずに，メモリ上で順次処理をし続けるストリームデータ処理を適用することで，データを受信した時に即座にそのデータの分析を行い，これによって実世界のリアルタイムな状況に即したサービス提供を行う事例が増えている [100,117].

ストリームデータ処理を含めたシステムの設計指針として，Nathan [83] は，データをバッチ的に処理するコールドパスと，リアルタイムに処理するホットパスの 2 つから成るラムダアーキテクチャを提案した．コールドパスはデータベースにストアした大量の蓄積デー

[1]https://www.ovice.in
[2]https://spatial.chat

タを対象とした分析や機械学習のモデル構築などの時間がかかる処理を行うことを想定しており，ホットパスはストリームデータ処理を想定している．さらに，Kreps [76] は，ストリームデータ処理のみによってシステムを構成するカッパアーキテクチャを提案しており，実装例 [9] も報告されている．

ストリームデータ処理は，継続的に発生する大量データを高スループットで遅延少なく処理できる一方で，デバイスからアップロードされるデータを変換・加工・集約する処理のパイプラインがストリームデータ処理基盤上で実行されることになり，デバイスからサービスにいたる複数の処理が密結合になりやすい [126]．結果として，デバイスからサービスにいたる一連のロジックが垂直統合的に開発されるとともに，サービスロジックが IoT デバイスの仕様に依存して実装されがちであり，サービスの修正やサービスを構成するロジックの再利用が難しくなりがちになる．このような垂直統合開発は IoT データを扱う際の一般的な傾向 [46] でもあり，これは昨今問題視されている，様々なロジックやサービスが相互に連携できない状態で分断して存在し全体最適が難しくなる「システムのサイロ化」の原因ともなる [134]．そのため，ストリームデータ処理基盤上で，デバイスとサービスの密結合を抑制した上で，複数の処理を柔軟に組み合わせて構成できるようにする必要がある．

1.2 研究の目的

本研究では，1.1 節で述べた 3 つの技術領域に対応し，大規模な空間データを保持・更新しつつインタラクティブに可視化することを実現するための 3 つの研究に取り組む．

1 つめは，空間を広域に可視化しようとする際に生じる，分散ノードからのデータ集約にかかる時間と負荷を解消することで，これまでは困難であった，時間経過ともに変化する大規模な空間の広域な可視化を可能にすることである．取り扱う空間の規模を大きくするためには，複数台のノードによってデータの保持や演算実行を分担する必要がある．この構成では空間データが複数のノードで分散保持することになるため，広い範囲を俯瞰的に可視化する際は，そのデータを保持する複数のノードからデータを集約する必要がある．これにかかる処理負荷や通信時間がボトルネックとなり，時間経過とともに変化する空間を広域に可視化することがこれまで実現できていない．そのため，分散ノード自身が P2P でデータを流通させる機構にもとづき，このボトルネックを解消する方式を提案する．

2 つめは，仮想空間上で臨場感のあるインタラクションやコミュニケーションを可能にすることを目的として，仮想空間上の注目地点を中心として，その地点からの物理的な事象が到達する範囲を近傍領域として定め，その領域を空間上の構造物を考慮しつつ決定す

るモデルを提案すること．また，変化する空間の構造や利用者の操作に遅延なく追従して領域の形状を変えていくことができる高速な計算方法を提案することである．1.1.2 節で述べたように，物理シミュレーションによって物理的な事象の伝達範囲は近似的に求めることができる一方で，物理シミュレーションの実行は計算負荷が高い．そこでこの研究では，物理モデルに立脚しつつも計算負荷の少ない軽量・高速に伝達範囲を近似する計算方法を提案する．

３つめは，仮想空間上のデータを IoT デバイスからアップロードされるデータによって更新することを主な用途として想定しつつ，1.1.3 節で述べたサイロ化の問題に対応するため，ストリームデータ処理基盤における処理のロジックを疎結合にした上で，それらを組み合わせることでストリームデータ処理のパイプラインを柔軟に構成する枠組みを提案することである．

そして，これら３つの研究により，図 1.1 で示した，大規模な空間データを保持しつつ，それを IoT デバイスからアップロードされる情報などに基づいて逐次更新しつつ，直感的なインタラクションを交えて広域に可視化する全体システムを実現することである．

1.3 対象とする空間データ

本研究が取り扱う対象とする空間データは２次元のラスタデータを想定している．そのため，本論文で述べるシステムでは，取り扱う空間を離散化した格子として表現し，２次元の配列データとして保持する．配列の要素には，空間の属性や状態を意味する値が格納され，各要素のサイズは高々数バイトに収まることを想定している．例えばゲームであれば，各要素のサイズを２バイトとし，上位８ビットを用いてその地点における仮想世界を構成する「土」や「草」などを区別する属性値を表現し，下位８ビットを用いて，温度，湿度，燃えるかどうか，プレイヤが通行可能かどうかといった，状態値や管理情報を表現することが考えられる．

仮想世界を，一辺の長さを 1m 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．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．また，仮想世界を構成する空間データのサイズは固定とは限らず，運用とともに拡大していくことを想定している．なお，仮想世界全体は一様な格子で現されるため，空間上の位置によるデータ量の偏りは存在せず，仮想世界のどこであっても面積あたりのデータ量は一定となる．

空間データは，その一部，あるいは全てが更新される場合がある．このため，空間を可視化する場合は，変化する空間の内容を極力すぐに反映して描画する必要がある．空間データは「事前に定めた規則による定期的な更新」,「システム利用者の空間への介入による更新」,「入力された IoT データの反映による更新」によって更新される場合がある．これらの更新処理についての詳細は 3 章で述べる．

1.4 提案手法と得られた成果の概要

１つめの研究の提案手法は，システムを構成する分散ノード同士が通信ネットワークを介して格子状に仮想的に P2P で接続した上で，各ノードが自身が保持する空間データを近接する周囲のノードに配信し，また，他のノードから受信した空間データをさらに他のノードへ仲介して送信することを繰り返すことで，空間の広域可視化に適した分布を生成する．空間データを収集する特別な代表ノードを設けるのではなく，システムを構成する全てのノードが同じ役割と機能を持って対等に通信することが特徴である．本方式を実装した上で性能検証実験を行い，ノードの数に応じたリソース消費負荷が一定内に抑えられること，代表ノードを用いる方式と比較して，単位時間当たりのクライアントからの空間データ取得リクエスト数が一定数を超えた時点で提案方式のほうが負荷が低くなる傾向を示した．

２つめの研究の提案手法は，利用者と仮想空間の直感的なインタラクションを目的として，注目地点の近傍領域モデルをポテンシャルによってモデル化する．具体的には空間上の注目地点を最大値，空間上に存在する構造物の壁面位置を最小値と拘束条件とした上で，その条件を満たすラプラス方程式の定常問題を解くことで近傍領域を生成する．また，偏微分方程式の近似解法である代用電荷法を応用することでこの問題を解き，リソースの少ない計算機でも計算負荷なく短時間で近傍領域を生成する方法を示した．また，実験によって計算に要する時間がポテンシャル 1 つあたり 50 ミリ秒であることを示し，遅延の少ないインタラクションを行う上で問題のない時間内で近傍領域を生成できることを確認した．

３つめの研究では，仮想空間のデータを IoT デバイスからアップロードされるセンサデータによって更新することを想定し，大量・高頻度かつ多様なセンサデータを柔軟に処理するための並列分散ストリームデータ処理基盤の枠組みを提唱する．提案手法では，並列分散ストリームデータ処理におけるキー付けされたストリーム（Keyed Stream）から成る基本処理フローの上に，オブジェクトとプラグインという抽象化された要素を組み合わせてフローを構成する枠組みを敷き，これによって並列分散ストリームデータ処理における処理フロー開発の難易度や，センサデータ仕様とサービスロジックの密結合化の低減を図る．

提案方式を実装した基盤の上でサービスを試作し，提案方式の効果を確認した．また，秒間最大 10 万件のデータアップロードを想定したワークロードによる実験で基本性能を確認した．

1.5 本論文の構成

本論文ではまず 2 章で,「大規模空間データの可視化」,「仮想空間上での近傍領域生成」,「ストリームデータ処理」についての前提知識を示す．次に，3 章にて大規模空間データの可視化を行うための分散システムの構成と，データの拡散レプリケーションによる負荷の分散方法について述べる．また，性能検証によって提案方式の効果を検証する．4 章では，空間上で注目地点の近傍領域のモデル化とその計算手法を提案し，5 章では，センサデータの変換処理を柔軟に行うためのストリームデータ処理基盤を提案する．最後に 6 章で本研究をまとめる．

第2章　前提知識

ここでは，1 章で述べた（１）広大な仮想空間を表現する大量の空間データを保持し，また，その一部や全てを更新し，それを複数の利用者に対して提供して可視化することを目的とした基盤.（２）仮想空間を対象として，空間をインタラクティブに探索したり，空間上で他の利用者とコミュニケーションをとるために臨場感のある直感的なインタラクションを行うための近傍領域.（３）IoT デバイスからアップロードされるセンサデータを処理するためのストリームデータ処理に関して，それぞれ前提となる知識を示す.

2.1　大規模空間データの可視化

2.1.1　並列計算方式の分類

Flynn [51] による分類を参考に，並列計算のタイプは SPSD（Single Program, Single Data stream），SPMD（Single Program, Multiple Data stream），MPSD（Multiple Program, Single Data stream），MPMD（Multiple Program, Multiple Data stream）の４種に分類できる（図 2.1）.

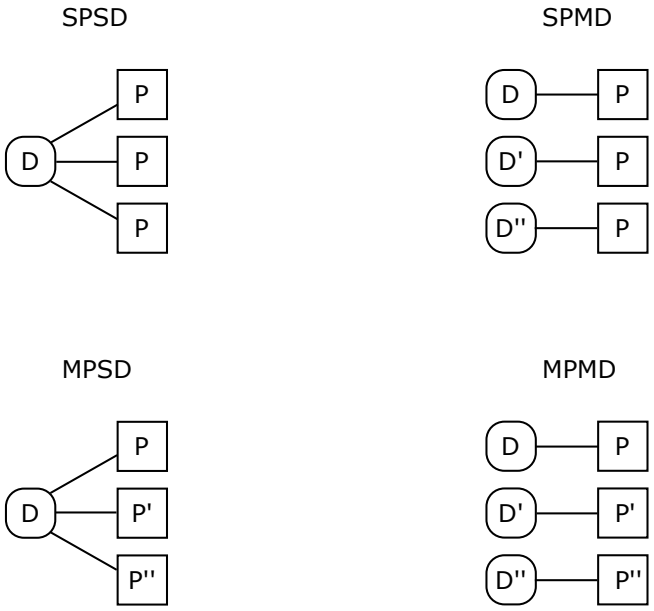
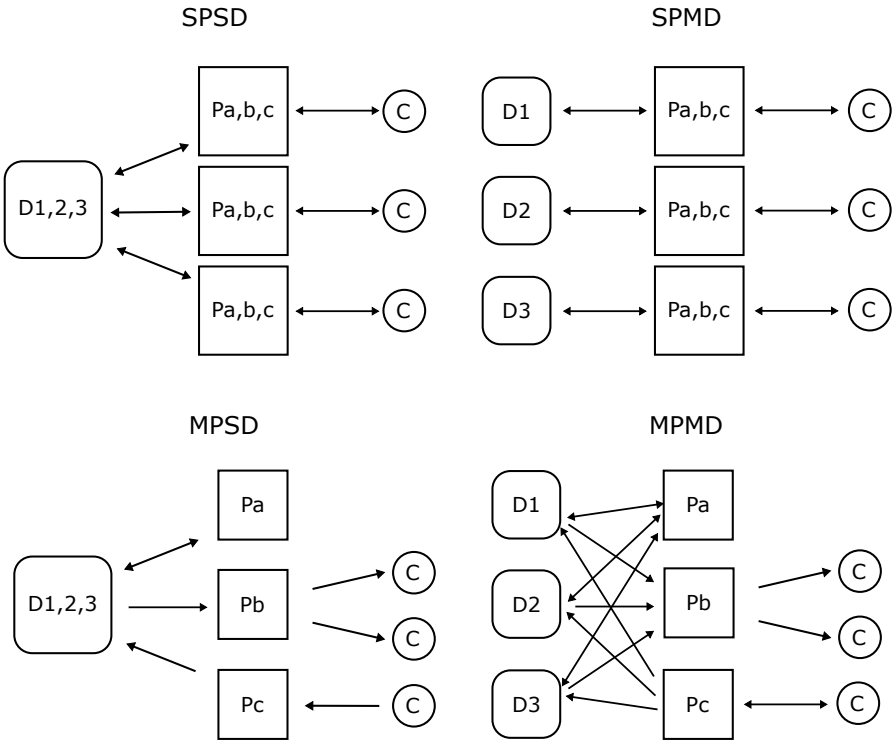


図 2.1: 並列計算方式の分類

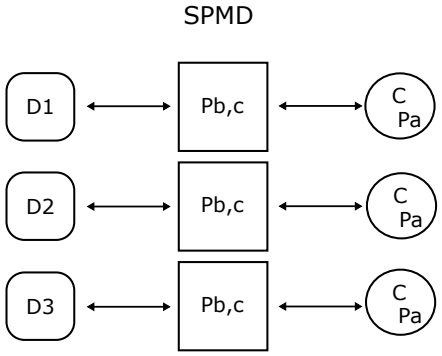
この分類の種別それぞれについて，大規模な空間データの可視化基盤を構成した例を図 2.2 に示す．本論文で述べる可視化基盤は，空間データを保持し，また，保持する空間データの一部もしくは全てを，事前に定めた更新規則によって定期的に更新する時間発展演算を施し，また，保持する空間データをクライアントからの要求に応じて提供したり更新することを想定している．この図において P_a，P_b，P_c はそれぞれ，P_a : 空間データに対して時間発展演算を実行し，データを更新するプロセス，P_b : クライアントからの要求に応じて空間データを読み取りクライアントへ提供するプロセス，P_c : クライアントからの要求に応じて空間データを更新するプロセスを表す．仮想世界の広さを大規模にするには，空間データを複数の部分データに分割し，それらを並列に取り扱う構成が必須となるため，空間データを分割せずに扱う SPSD と MPSD は非現実的となる．なお，3 章で述べる可視化基盤では，システムの構成を単純にし保守性を高めるため，システムの構成要素に均質性を持たせることを指向している．このため，システムの構成要素が異なる役割を持つ MPMD は均質性の要件を満たさず，結果として 3 章で述べる可視化基盤の構成としては，SPMD が最も適切となる．



D1,2,3: 空間データ
Pa: 空間データの時間発展プロセス
Pb: クライアントへの空間データ提供プロセス
Pc: クライアントからの要求に応じた空間データ更新プロセス
C: クライアント

図 2.2: 分散可視化基盤の構成例

なお，図 2.2 で示した構成は，空間の時間発展計算処理をサーバ側で行う想定である．これに対して，図 2.3 に示すように，この処理をクライアント側で実行する構成も考えられる．これは，サーバにアクセスしたクライアントがサーバから空間データを受信し，その後はクライアント側で空間の更新処理を行い，格子データを更新しつつその結果を定期的にサーバに送信する方式である．例えばゲームへの応用を想定した時，プレイヤが表示している仮想空間の領域の更新処理を，そのプレイヤのクライアント PC で行う構成が挙げられる．この場合，サーバ側に必要となる計算処理負荷は軽減するが，クライアントに対して更新対象データを排他的に割り当てる機構が複雑となりがちである．また，空間データの全領域分の更新処理をクライアントに割り付けられるわけではなく，空間上で時間発展の進行に大きな差が生じる可能性がある．このため，クライアント側で更新処理を行う構成は，このような制約を許容できるアプリケーションの場合にのみ適用できる構成である．



D1,2,3: 空間データ
Pa: 空間データの時間発展プロセス
Pb: クライアントへの空間データ提供プロセス
Pc: クライアントからの要求に応じた空間データ更新プロセス
C: クライアント

図 2.3: 分散可視化基盤の構成例（空間の時間発展計算をクライアント側で実行する場合）

2.1.2 P2P における情報伝達方式

本論で述べる可視化基盤ではシステムを構成する分散ノードを P2P（Peer to Peer）で接続する．P2P とは，分散システムを構成するノード（ピア）がネットワークを介して相互に接続するアーキテクチャである．P2P において，ノード同士の情報伝達方式には幾つかの基本的な方式が存在している．

フラッディング

最も基本的なものとして，ノードが近接するノードに対してデータを送信し，それを受け取ったノードはさらに近接するノードに対してそのデータを送信することを繰り返すフラッディング方式が存在している．フラッディング方式は送信されるデータの量が級数的に増加するため，データを伝達する回数（ホップ数）や時間に制限をかける必要がある．

ゴシッププロトコル・エピデミックプロトコル

ゴシッププロトコル [19] はエピデミックプロトコルとも呼ばれ，フラッディング方式のようにノードが全ての近接ノードへデータを送信するのではなく，ランダムに選んだ相手に対して確率的にデータを送信する方法である．フラッディング方式と比較してデータの通信量を抑えながらシステム全体に情報を広められる利点があり，データベースのレプリケー

ションや故障検知 [92, 109]，携帯端末間での情報拡散 [99]，アドホックルーティング [59] などで多く利用されている．

格子状の P2P の平均経路長

P2P のノード同士が格子状に仮想的に配置され，ネットワーク接続されているシステムでは，システムを構成する 2 ノード間の経路距離の平均（ASP: Average shortest path length）が解析的に求まっている [74]．また，ノードがフラッディング方式によってデータを他のノードに伝達する場合，それに要する平均時間は ASP から見積もることができる．D 個のノードが格子状に接続されているシステムにおける ASP のオーダは $\sqrt{D}$ である（図 2.4 左）．また，このネットワークに対して近接ノードと接続されたリンクを，ある確率（$p = 0.1$ 程度）で異なるノードへのリンクへ張り替えることによって生成する Watts-Strogatz モデル（WS モデル）[106] に修正を加えた，リンクを張り替えるのではなく追加していく修正 WS モデルによってノードネットワークを構築することで，ASP のオーダは $logD$ となる（図 2.4 右）．

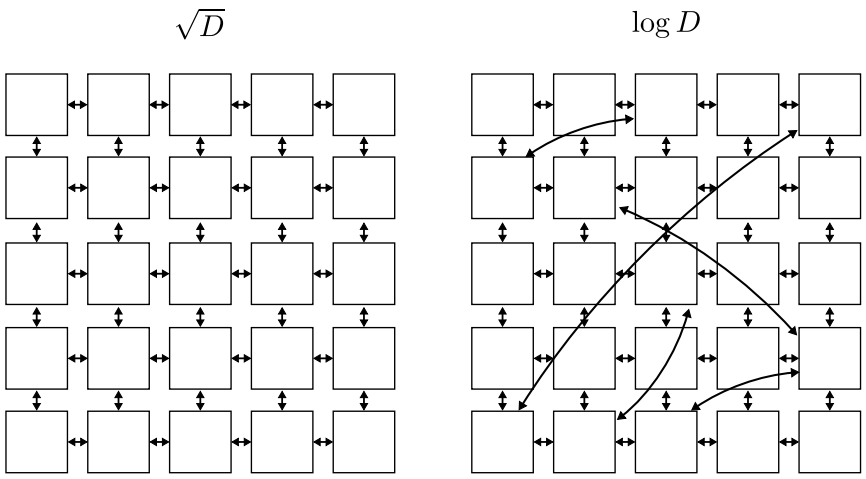


図 2.4: 格子状のノードネットワークの平均経路長のオーダー

2.1.3 ハイパフォーマンスコンピューティング領域における研究

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) [25] 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) [24] 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特徴がある．

In-Situ 可視化

スーパーコンピュータを用いて，科学シミュレーションを始めとした大規模シミュレーション行った結果の可視化方法として，大量の計算結果をいったんストレージに保存して可視化するのではなく，シミュレーション実行時に可視化のための比較的軽量なデータを生成し，シミュレーションの実行と可視化を同時に行う In-Situ 可視化がよく行われる [132]．川村 [72] は，可視化データを粒子によって表現することで，計算に要する時間やデータ転送量を削減しするとともに，また，可視化結果を閲覧するユーザ PC と，スーパーコンピュータを連携し，これらによって高速な In-Situ 可視化を実現する方法を提案している．

2.1.4 クラウドコンピューティング領域における研究

分散データベース基盤

Geographic Information System（GIS）分野では空間情報を扱うことに特化した，あるいは空間情報を扱うための拡張機能を有する分散データベース基盤が多く存在している．MySQL [17], PostgreSQL [21] などの Relational Databse 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，MongoDB [14], Redis [22] 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物は多数存在する．MD-HBase [88] は多次元のインデックスによって空間情報の検索をより効率的に行う分散 KVS である．DISTIL [91] 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上で，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い合わ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．GeoMesa [64], GeoSpark [111] は Accumulo [2], HBase [6], Bigtable [42], Cassandra [78] といった分散データベース上で大規模な空間情報の解析を行うシステムである．これらのシステムは，いずれもデータがパーティション，シャード，リージョン，バケット，タブ

レットといった呼び名の単位で分割されており，大量の時空間データを効率よく保持・検索・変換するためのスケーラビリティを有している．

ラスタデータ解析基盤

Guan らは，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 general-purpose parallel raster processing programming library (gRPL) [55] を提案している．gPRL が想定しているラスタ処理は，例えば農業・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．同様の目的のための，高速なラスタ処理方式は [56, 79, 108, 114, 115] で提案されている．また，[116] では，GPU を搭載した計算機群で構成したクラスタによる大規模化の可能性についても示唆している．

地理データ可視化システム

大規模な空間データの可視化する際には，可視化する際の縮尺に応じた異なる詳細度（LOD: Level Of Deatil）のデータを複数保持するデータピラミッドを生成する場合が多い．Guo ら [58] は，地理空間に対してパッチ状に存在するラスタデータと，そこに対する表示要求（クエリ）の MBR を考慮して，ピラミッドデータの生成時間を短縮する方法が示されている．[69, 80, 112] では，惑星の地形データを３次元表示する統合システムについて述べられており，４分木アルゴリズムによる視野領域を考慮したデータピラミッドの効率的なメモリ読み込み，割当て方法が示されている．バーチャル地球儀／地図として知られている Google earth[1]/Google Map[2] は，衛星画像を始めとした多様な地理画像や地図データを，大規模クラスタ管理システムである Borg [103]，分散データベース基盤の BigTable [42] や Spanner [44] といったシステムを組み合わせて，定期的にダウンサンプリングしてピラミッドデータして保持し，それらをインターネットを介してクライアントに提供することで，地球全域にわたって地形や都市の様子を詳細に可視化をすることを可能にしている．しかし，既存の地理データ可視化システムは，データを頻繁に更新することを想定していない．比較的データの更新頻度の高い Google earth であっても，最も短い更新間隔を持つ気象データでも 1 時間ごとの更新を想定している [52].

[1] https://www.google.co.jp/intl/ja/earth/
[2] https://www.google.co.jp/maps/?hl=ja

大規模多人数同時参加型オンラインゲーム (MMORPG) のための分散システム

MMORPG (Massively Multiplayer Online Role-Playing Game: 大規模多人数同時参加型オンラインロールプレイングゲーム) 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上で，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，あるいは P2P ノード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 [66, 68, 104, 125, 137]．[137] 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，プレイ空間の領域ベースの分割方式が効率的である．また，多くの MMORPG のための分散アーキテクチャでは，図 2.5 に示すような，領域ベースで個別に駆動する各サーバ（ノード）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，共有するための，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いる．

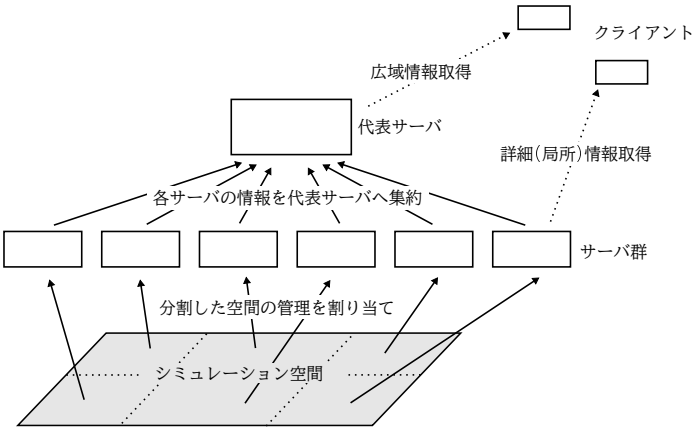


図 2.5: 代表サーバ方式による MMORPG システムの構成例

2.2 仮想空間上での近傍領域生成

2.2.1 物理シミュレーションに関する技術や手法

仮想空間上で臨場感のあるインタラクションを行うために，仮想空間上で何かしらの物理的な事象の伝達や分布を再現する物理シミュレーションに関する技術や手法について述べる．

レイベース法

仮想空間上の特定地点から音響の伝播を再現する手法として，レイベース法 [96] が存在している．レイベース法はコンピュータグラフィックスのレイトレーシング法と同様の手法であり，音源から多数の粒子を放射して直進させ，粒子が壁面に衝突したら反射して方向を変えた上で再度直進することを繰り返すことで，空間構造内の音響を再現する手法である．後述する波動ベース法と比較すると計算負荷が少ない特徴がある．しかし，粒子の進行を再現するために反復的な計算を何度も繰り返す必要があり，その分の計算時間と負荷はかかる．

波動ベース法

波動ベース法 [93] はレイベース法同様に，仮想空間上の音響伝播を再現する手法である．波動ベース法は格子で表現した空間上で FDTD（Finite Difference Time Method）を始めとした数値計算を用いることで，格子毎の音圧値の変動を計算する方式であり，格子のサイズを細かくすることでレイベース法よりも精度の高い，また複雑な空間形状にも対応した音の反響シミュレーションが可能となる．しかし大量の格子を対象として計算を繰り返す必要があり，計算負荷が高くなる．

ラプラス方程式とその解

ラプラス方程式（式 2.1）は空間の 2 階微分項からなる偏微分方程式である．

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.1}$$

この式は時間に依存せず，境界値を満たすような定常分布解が一意に得られる．この定常分布解はポテンシャル，あるいは調和関数とも呼ばれ，非圧縮性流体の円柱周りの速度の分布や，電荷点周りの電位の分布，熱媒質中の熱の分布などの様々な物理的な事象の定常状態を表現する．

ラプラス方程式の境界値問題はポテンシャル問題とも呼ばれている．ポテンシャル問題は，解析的に解けることは極めて稀であり，ほとんどの場合に数値計算を用いて解くことになる．格子状に離散化した空間における点 i, j における式 2.1 の u を $U_{i,j}$ と記述した上で簡単な計算を行うと，式 2.1 は次のように書き下すことができる．

$$U_{i,j} = \frac{1}{4}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}) \tag{2.2}$$

空間を n 等分して離散化した場合は，式 2.2 の線形方程式が $(n-1)^2$ 個得られることになり，これを連立して解くことで定常分布解を得ることができる．空間の分割数 n が多くなると連立方程式の次元が高まり，求解に要する時間が長くなる．

代用電荷法

ラプラス方程式の境界値問題を近似的に解く手法として代用電荷法 [33,70,71] が存在している．代用電荷法は，ポテンシャル分布を計算する空間上に，幾つかの電荷点と，その電荷点から発生する電位の量を拘束する標本点を配置し，標本点で指定した値を満たすようなポテンシャル分布を近似的に計算する．

代用電荷法によるポテンシャルの生成手順を次に示す．

1. 図 2.6 に示すように，標本点（Collocation points）$\{\boldsymbol{x_i}; i = 1, 2, \cdots, n\}$ をポテンシャルを計算する領域 Ω に沿って配置する．
2. 電荷点（points）$\{\boldsymbol{y_i}; i = 1, 2, \cdots, n\}$ を Ω の外側に，標本点の近くに配置する．
3. 標本点 $\{\boldsymbol{y_i}; i = 1, 2, \cdots, n\}$ における定数 $\{\boldsymbol{b_i}; i = 1, 2, \cdots, n\}$ を決定する．
4. 式 2.4 を解くことで Q_i を求め，その値を用いて，式 2.3 によって位置 x におけるポテンシャル $U(\boldsymbol{x})$ が得られる．

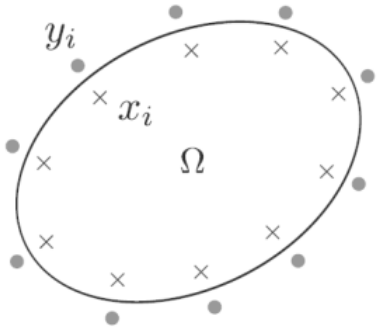


図 2.6: 代用電荷法 x_i が標本点（Collocation points）,y_i が電荷点（Charge points）

$$U(\boldsymbol{x}) = \sum_{i=1}^{n} Q_i \log|\boldsymbol{x} - \boldsymbol{y_i}.| \tag{2.3}$$

$$\boldsymbol{mQ} = \boldsymbol{b}. \tag{2.4}$$

$$\boldsymbol{m} = \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n} \end{pmatrix} \tag{2.5}$$

$$m_{j,k} = \log|\boldsymbol{x_j} - \boldsymbol{y_k}| \tag{2.6}$$

$$\boldsymbol{Q} = \begin{pmatrix} Q_1 & Q_2 & \cdots & Q_n \end{pmatrix}^{\mathrm{T}} \tag{2.7}$$

$$\boldsymbol{b} = \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix}^{\mathrm{T}} \tag{2.8}$$

簡単のために，1 次元空間上のポテンシャルを代用電荷法を用いて求める計算例を示す．図 2.7 に示すように，解析空間上の 2 点（$x_1 = 2, x_2 = 8$）をポテンシャル境界位置として定めて標本点を置き，この点における境界値 b を，それぞれ $b_1 = 2, b_2 = 4$ として定める．

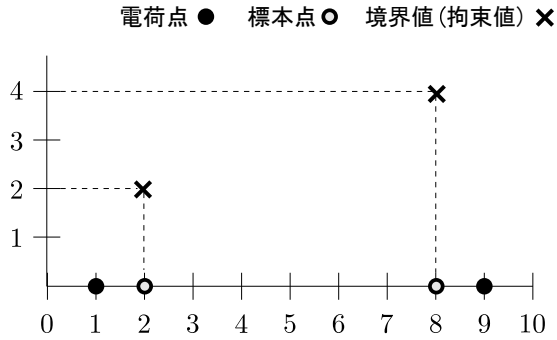


図 2.7: 1 次元空間上への電荷点と標本点の配置と境界値

これらの値から，式 2.3 で表現されるポテンシャルを具体的に求めると

$$U(x) = Q_1 \log|x - 1| + Q_2 \log|x - 9|$$

となる．この式の Q_1 と Q_2 は式 2.4 によって表現される次の連立方程式

$$\begin{pmatrix} \log|2-1| & \log|1-8| \\ \log|8-1| & \log|8-9| \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} \tag{2.9}$$

を解くことで，$Q_1 \approx 2.055593369, Q_2 \approx 1.027796684$ が得られる．これらの値を式 2.7 に代入すると，目的のポテンシャルが得られる．

つまり代用電荷法は，標本点で定めた境界値を拘束条件としながら，電荷点を中心として分布する対数関数の重ね合わせによってラプラス方程式の境界値問題を近似的に解く方式である．この様子を図 2.8 に示す．

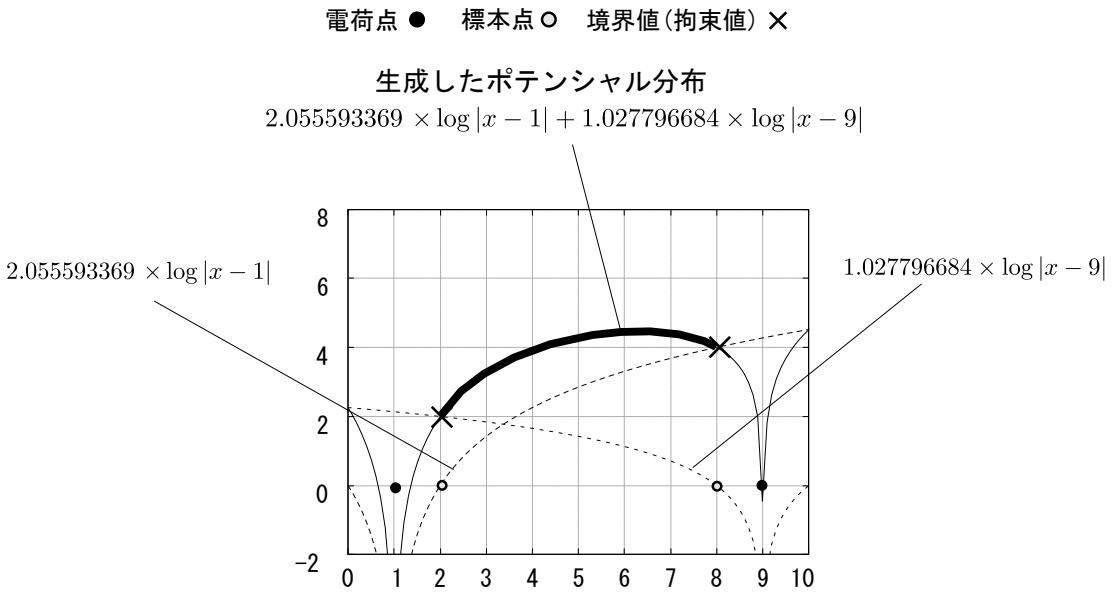


図 2.8: 代用電荷法によって 1 次元空間上に生成したポテンシャルの例

代用電荷法は解くべき連立方程式の元数を標本点の数にまで減らすことができるため，計算時間を劇的に向上させることができる．

2.2.2　非言語情報によるインタラクション制御モデル

ゲームやコミュニケーションツールにおいて，遠隔の利用者同士が仮想空間上で何かしらの共同作業を行おうとした際，利用者同士が円滑に意思疎通できることが重要となる．現実世界のコミュニケーションでは，相手の声や匂いや体温などの非言語情報が，円滑な意思疎通に重要な役割を果たしている [32]．このため，仮想空間上で現実世界と同様に非言語情報を伝達する枠組みとして，利用者が操作するキャラクタなどの特定の地点を中心とした近傍領域を生成し，その領域内に非言語情報が伝達する範囲を限定するなどの制御を行う方式が多く提案されている．以下にそれらの例を挙げる．

オーラ，フォーカス，ニンバス

Fahlén [38, 50]，Greenhalgh [53] は，仮想空間において，キャラクタの周囲を取り囲むように幾何学的な形状を定義し，これをもとに他のキャラクタとのコミュニケーションチャ

ネルを確立する方式が提案されている．この幾何学形状は「オーラ」,「フォーカス」,「ニンバス」という分類に分かれている．オーラは全てのキャラクタに対等に存在する近傍領域であり，オーラ同士の重なりを判定することで，キャラクタ同士，あるいは複数のキャラクタが集まったグループ間でインタラクションに関する制御を行う方式である．一方，フォーカスとニンバスはキャラクタ同士の非対称なインタラクションを制御するために用いるものであり，キャラクタの向いている方向などによって指向性を持たせたフォーカスが，特定のキャラクタのニンバスに重なっていることをもってインタラクションを制御する．

アウェアネス制御

仮想空間上での音声チャットフレームワークである Ravitas [110] では，音源からの距離に応じて音が聴こえる範囲を定め，また，一定の距離の範囲内にいるキャラクタ同士の音声を他より鮮明にする制御が提案されている．[127,128] では，仮想空間上のキャラクタからの位置関係にもとづいて他のキャラクタに対して提示する情報量を制御するアウェアネス制御モデルが提唱されている．

2.3 ストリームデータ処理

2.3.1 ストリームデータ処理の基本

ストリームデータ処理とは，継続的に発生するデータをリアルタイムに処理する技術である．1992 年に，データベースを継続的に監視し，条件を満たしたことを検出する連続的な問合せ（Continuous Query）の概念が Tapestry [101] によって提唱され，2000 年代初頭には TelegraphCQ [41]，STREAM [35] などのストリームデータ処理を行う専用システムが開発された．2015 年頃からは Spark [8]，Storm [67], Flink [40] に代表されるビッグデータに対応したクラウドベースの並列分散ストリームデータ処理基盤が台頭している [60].

ストリームデータ処理システムの最も大きな特徴は，発生したデータをいったんデータベースに蓄積した上で，後でそれを参照取得して使うのではなく，発生したデータをそのデータベースに蓄積せずその都度メモリ上で分析することで，遅延の少ない処理を実現するところにある（図 2.9）．IoT デバイスは多くの場合，センサデータの値を継続的にサーバにアップロードする．これらのセンサデータをストリームデータ処理基盤の上で処理することで，例えばセンサデータの値を監視して閾値を上回ったら即座にアラートを発呼するといった，実世界の状況に連動したサービスの実現が可能となる．

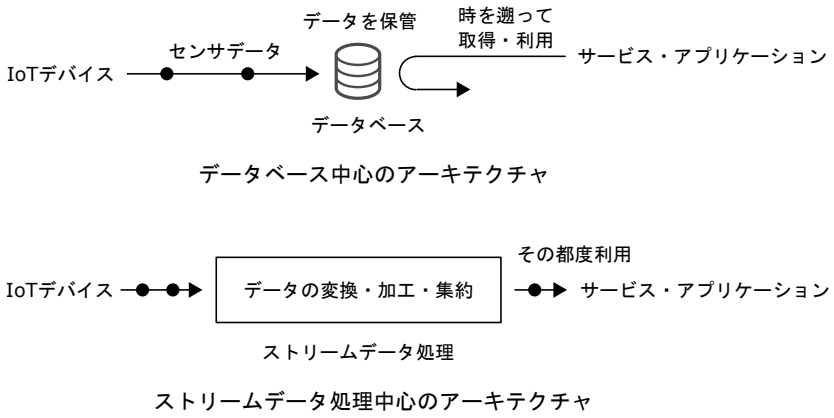


図 2.9: データベースを用いたシステムと，ストリームデータ処理を用いたシステム

ストリームデータ処理の処理フローは，データが入力されてくるポイント（Source）から発生するデータに対して繰り返し演算子（Operator）を作用させることでデータの変換や加工，分析を行い，結果をデータの出力ポイント（Sink）に出力する流れとなる（図 2.10）.

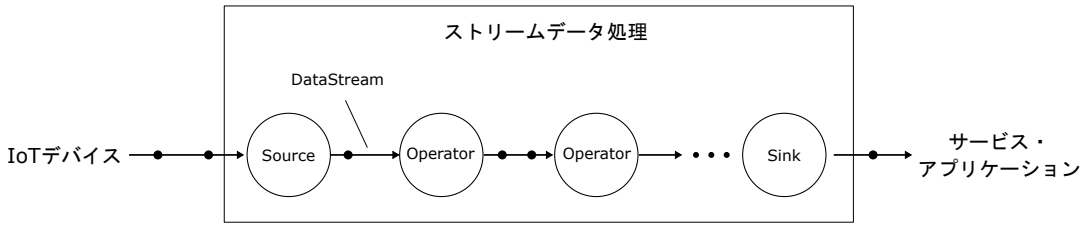


図 2.10: ストリームデータ処理の流れ

ストリームデータ処理における時間の概念

ストリームデータ処理では図 2.11 に示すように，継続的に発生するデータに対して特定の時間幅（タイムウィンドウ）を定め，そのタイムウィンドウに含まれるデータを処理の対象として扱う場合が多い．タイムウィンドウの制御方式には様々なものが存在しており，代表的な物はウィンドウの幅と前進間隔が等しい「Thumbling Window」や ウィンドウのサイズに対して前進間隔が小さい「Sliding Window」がある．

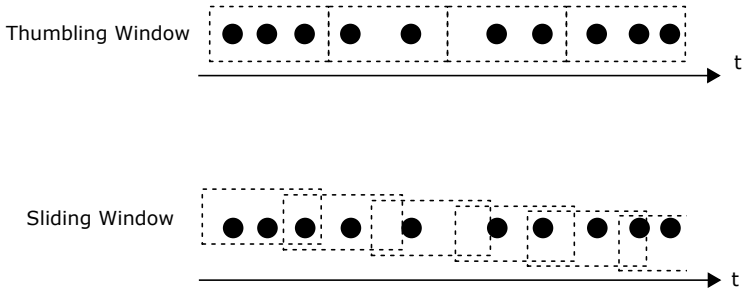


図 2.11: ストリームデータ処理における Event Time と Processing Time の関係

タイムウィンドウを制御するにあたって，その基準となる時刻には幾つかの種別が存在しており，代表的な物は「Event Time」と「Processing Time」である．Event Time はデータが発生した時点で IoT デバイスなどのデータの発生源によって付与されるものである．これに対して Processing Time は，そのデータを処理する時刻であり，多くの場合ストリームデータ処理を実行する計算機のマシン時刻となる．

多くの場合，ストリームデータ処理を行う対象のデータはネットワークを介して届けられるため，そのデータが届くまでには遅延が生じ，Event Time と Processing Time の間には図 2.12 に示すような歪み（Skew）が常に存在する．

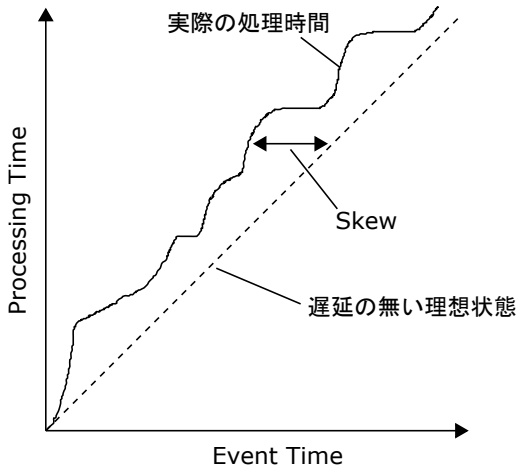


図 2.12: ストリームデータ処理におけるタイムウィンドウの例

タイムウィンドウを Event Time にもとづいて制御する場合，この歪みを考慮して，Event Time ベースでどの時点までのデータを処理したことと見なすのかを目安として管理する

ウォーターマーク機構が必要となる．その上で，ウォーターマークベースでウィンドウを閉じて，ウィンドウ内のデータを対象とした集計処理を実行するトリガー機構も必要となる．先に挙げたストリームデータ処理基盤にはこれらの機構が備わっている．また，到着するデータの統計的な遅延傾向に応じてウォーターマークの生成タイミングを適応的に調整する方式 [89] も提案されている．

ステートフルなストリームデータ処理

ステートフルなストリームデータ処理では Operator が処理を実行した結果を状態値（State）として保持しておき，次回にデータが到着して処理を実行する際に参照することができる．例えば機器の温度を監視する処理の場合，アップロードされてくる計測温度が閾値以上か以下であるかの判定はステートフルでないステートレスなストリームデータ処理でも実現可能である．これに対し，計測温度の変化傾向を捉えて分析を行う場合は，過去のデータを参照できる，ステートフルなストリームデータ処理が必要となる．

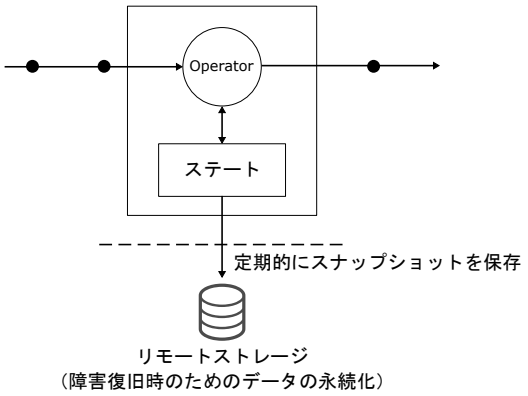


図 2.13: ステートフルなストリームデータ処理

図 2.13 にステートフルなストリームデータ処理のためのシステム構成の典型例を示す．ステートは，メモリや，SSD（Solid State Drive）のような高速ストレージでの利用を想定した LevelDB [13]，RockDB [20] 等により，その Operator が動作しているマシンのローカルに保持される．ステートの内容は定期的にリモートに置いた HDFS [5] 等のデータベースにスナップショットとして書き出されて永続化され，ストリームデータ処理基盤が障害停止して再起動した後の復旧時に利用される．

2.3.2 並列分散ストリームデータ処理

並列分散ストリームデータ処理は複数台のメニーコア CPU を搭載したマシンによって，大量のデータを対象としたストリームデータ処理をを分散して実行する技術である．並列分散ストリームデータ処理では図 2.14 のように，サービスの開発者が定義した処理フローが実行時は適宜分散し，複数台のマシンに割り当てられて並列に実行される．

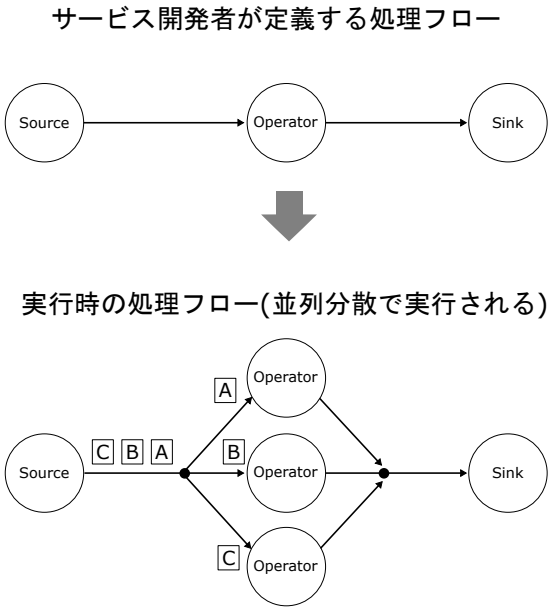


図 2.14: ストリームデータの並列分散処理

Apache Flink

Apache Flink [4] は分散スナップショット機構による障害発生時の一貫性のあるステート復旧 [39] や，高スループットと低レイテンシ [43] を特徴とするステートフルなストリームデータ処理を行う OSS の並列分散ストリームデータ処理基盤である．図 2.15 に Flink クラスタの上に処理フローが分散配備された例を具体的に示す．Flink は サービス開発者が定義した処理フロー（Task）を元に複数の SubTask を生成し，それらをワーカノードである TaskManager が保持する TaskSlot に展開して並列的に実行する．通常，1 台のマシンに 1 つの TaskManager を動作させ，TaskSlot の数は TaskManager が動作しているマシンの CPU コア数と同等にする．図 2.15 の例では 2 つの TaskSlot を保持する TaskManager を 2 つ動作させている様子を示している．

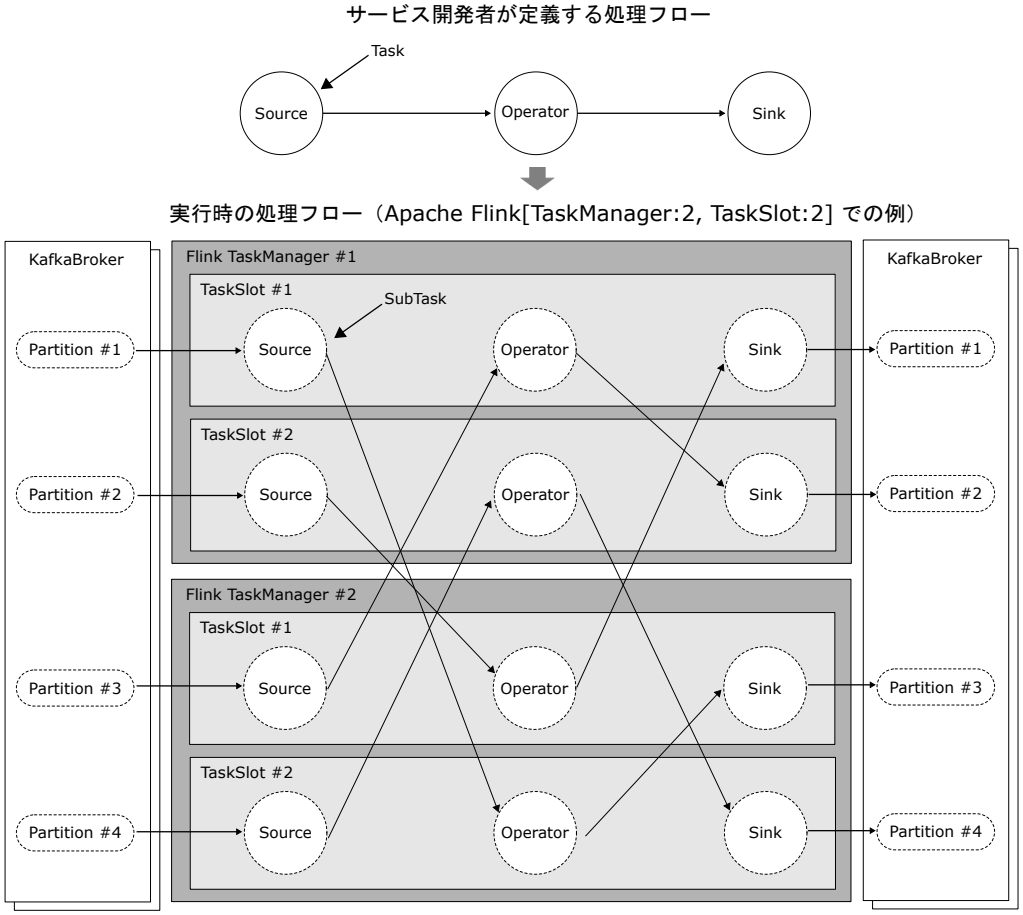


図 2.15: ストリームデータ処理の分散配備の具体例（Apache Flink と Apache Kafka を利用した場合）

なお，並列分散ストリームデータ処理によって大量のデータを扱う場合，ストリームデータ処理基盤に対するデータの入力（Source）や出力（Sink）も適切に並列化する必要があり，これに合わせて入出力のチャネルも並列化する必要がある．これを可能にするため，Apache Kafka [7] や Kinesis Data Stream [1] に代表される分散メッセージングシステムを介してデータの入出力を行う場合が多く，メッセージを送受信するためのチャネルを Kafka の場合は Partition，Kinesis の場合は Shard といった単位に分散して並列に置き，Source と Sink とデータの送受信関係を対応付ける構成がとられる．図 2.15 では，Flink クラスタのデータ入出力チャネルに Kafka を利用する構成例を示している．

並列分散ストリームデータに対する演算

並列分散ストリームデータに対する処理には幾つかの型が存在する．図 2.16 に Flink で提供されている Operator を例に典型的な物を示す．図 2.16 の 1 に示した Map Operator は，

入力されたデータ全てに対してそれぞれ演算を施すものである．ここでは入力されたデータの value を 2 倍にする例を示している．2 に示した Filter Operator は特定の条件を満たすデータ以外を除去するものである．ここでは value が 1 以下のデータを除去している例を示している．3 に示した KeyBy Operator は，データを指定したキー毎に分類して仕分けるものである．分類後のデータストリームはキー付けされたストリーム（Keyed Stream）となる．ここではデータの中の type をキーに指定して仕分け，Keyed Stream を出力する例を示している．4 に示した Window Operator は先述したタイムウィンドウを生成するものであり，5 に示した Window Appy Operator は生成したウィンドウを対象として，その中に存在するデータに対して一括して演算を施すものであり，ここではウィンドウ内のデータの value を加算する例を示している．

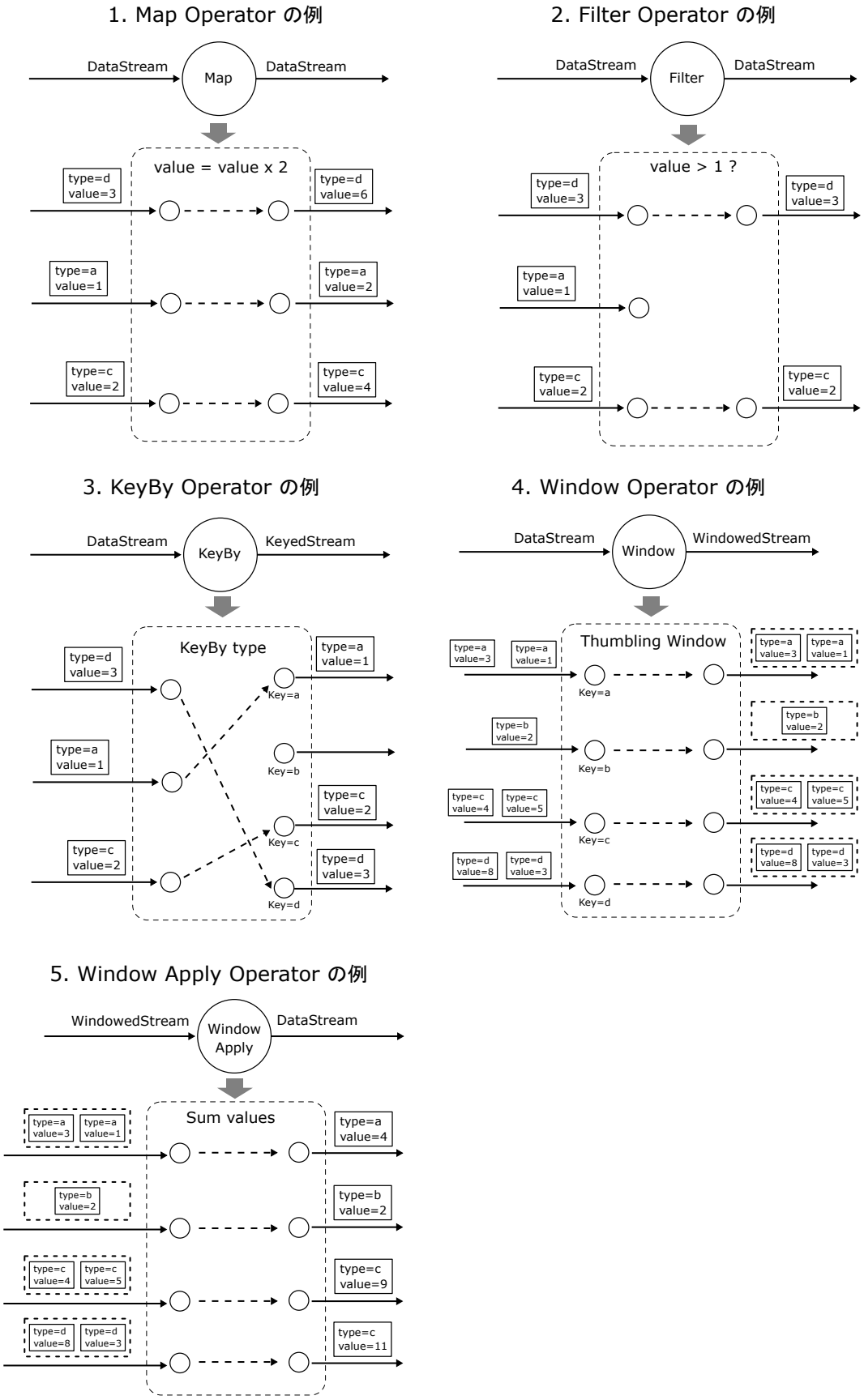


図 2.16: 並列分散ストリームデータに対する演算の例

第3章　大規模空間データの可視化を目的とした分散システム

3.1　背景と目的

3.1.1　空間の大規模化が求められる背景

大規模な空間データを保持・更新しつつそれを広域的に可視化するニーズが高まっている．本節では２つの使用事例にそって具体例を説明する．

広大なプレイフィールドを持つオンラインゲーム

物理，化学反応や動植物の生殖といった自然現象，あるいは人口増減や地価変動などの都市動態が模擬的に再現された仮想世界の中で，多数のプレイヤが地形生成や探索，構造物の構築，都市育成を楽しむサンドボックス型ゲームが台頭している．このようなゲームの古典的な例として，SimCity™[1]がある．SimCity™ は，商業施設や工業施設，住居，水道や電線といった要素によって構成される仮想都市の育成を楽しむゲームであり，まちづくりや都市計画のための教育用途にも用いられている [29,36,73,77,84]．SimCity™ では，都市の構成要素の状態を，近隣に配置された他の要素との相互作用にもとづいて更新することを時間経過とともに繰り返すことで都市の発展がシミュレートされる．これと同様のシミュレーション方式を持つ他のサンドボックス型ゲームの例として，Minecraft™[2]がある．Minecraft™ 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サンドボックスゲームである [37,48,85,86,90]．Minecraft™ 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．また，WorldBox™[3]は，雷や酸性雨，火山噴火などの天変地異が生じる惑星上に人間や羊，狼，エルフ，小人などの種族を放ち，その営みを鑑賞するゲームであり．これも同様に,「水」,「土」,「森」といった属性を持つパーツが近接のパーツと相互に影響を及ぼしながら自然現象をシミュ

[1] https://www.ea.com/games/simcity
[2] https://www.minecraft.net/
[3] https://www.superworldbox.com/

レートする．いずれのゲームも，ゲーム空間が何かの属性を持つ要素の集合で構成されること，また，それらの構成要素が近隣の構成要素と特定の規則にもとづいて局所的な相互作用を繰り返し状態を遷移させていくことで仮想世界の時間発展がシミュレートされる点は共通している．このようなシミュレーションは Cell Automaton(CA) [107] と見なすことができ，本基盤は，CA に代表される，空間を構成する要素同士の局所的な相互作用を繰り返し計算することによる系の時間発展シミュレーション実行をしつつ，そのシミュレーション空間を広域に可視化することを目的としている．

SimCity™，MineCraft™，WordBox™ は，現状ではシミュレーションを実行できるゲーム空間の広さに制限が存在する．MineCraft™ ではシミュレーションが実行されるゲーム空間上の範囲を，プレイヤが操作するキャラクタを中心とした距離を設定値として定める．その距離は計算機のリソースによって規定され，高々数百ブロック分（1 ブロックが 1 辺 1 メートル四方のスケール）が上限となり，その範囲外に位置する空間のシミュレーションは実行されず，時間進行が止まっている状態となる．SimCity™ にはこのようなシミュレーション距離は存在せず，ゲーム空間全域にわたってシミュレーションが実行される．しかし，シミュレーション対象の都市の面積は高々数キロ四方であり，対象とするゲーム空間を広くとることができない．WorldBox™ も同様である．他のサンドボックス型のゲームも同様に，計算機リソースの制約から，時間発展シミュレーションを実行できるゲーム空間の広さには制約が存在する．もし地球規模の面積（1 辺 1 メートル換算で約 5.101×10^8 個の格子）と同等の広さを持つ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，創発現象を始めとした多様な現象を再現できる．また，ゲーム空間を詳細表示（ミクロな視点）と広域表示（マクロな視点）をシームレスに行き来しながら可視化できれば，空間内で様々なスケールで生じている多様な現象を確認することができ，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．このようなゲームのイメージを図**??**に示す．

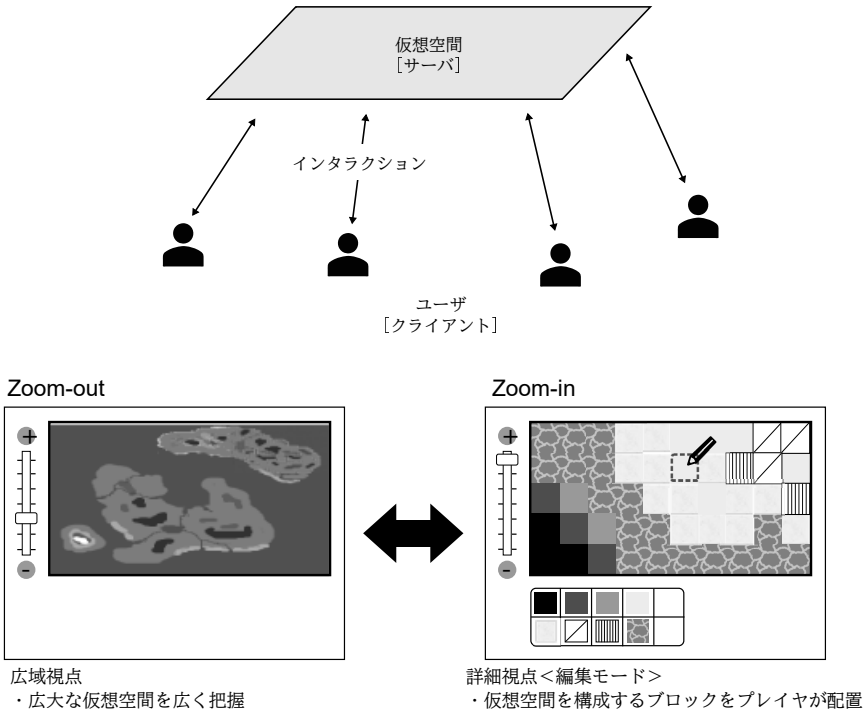


図 3.1: 広大なプレイフィールドを持つオンラインゲーム基盤のイメージ

都市交通の把握

都市を走行する大量の車両を対象とした交通流の把握のために本基盤を用いることが考えられる．IoT と自動車技術の発展により，走行中の車両に関する位置や速度といった状態値を継続的にインターネット上のサーバにアップロードし続けることができるコネクテッドカーの台数が増えている．富士経済によると，2019 年のコネクテッドカーの新車販売台数は 3,120 万台であり，2035 年には 9,420 万台に達すると予測されている [135]．本基盤は，これらの大量の車両からアップロードされるデータを地図上にマッピングし，さらにそれを広域に可視化することを目的としている．また，センサデータを地図上に重畳表示するだけでなく，車両の挙動や交通流を再現するシミュレーションモデルを基盤上で持ち，それをセンサデータをもとに補正しつつ時間発展計算を行うことで渋滞予測を行うような場合にも本基盤を適用できると考えている．例えば [34, 118] では，センサによって取得した車両の位置に基づき，道路区分における車両の存在密度を示す量をフェロモンとして定義し，近接道路区分間でフェロモンの拡散過程をシミュレーションすることで，渋滞の短期的な予測を行う方式が示されている．この方式は，道路区分ごとに存在する道路サーバが，その道路に存在する車両のセンサデータを基にその地点におけるフェロモン量を計算しつつ，近接する道路サーバとのみ情報を交換することでシミュレーションを自律的に進める．

本基盤ではこのような方式の計算を行うことも想定している．

渋滞予測状況にもとづいて，それを緩和する施策をシミュレーションによって検討する場合，狭い区域を対象にシミュレーションを行っても，その施策は局所最適に陥っており，その区域を含んだより広い領域の交通流が逆に滞る可能性がある．したがって，都市交通問題の根本的対策のためには，例えば東京都であれば 23 区全域といった範囲の広域エリアを対象として，より多数の車両から得られるセンサデータをもとにシミュレーションを行う必要がある．さらにシミュレーションの結果を可視化する際には，広域的な表示で広いエリアの交通量分布を把握することと，限定されたエリアを表示して詳細な交通流を把握することの両方が必要であり，この 2 つの表示をシームレスに切り替えられることが望ましい．これを実現するシステムのイメージを図 3.2 に示す．

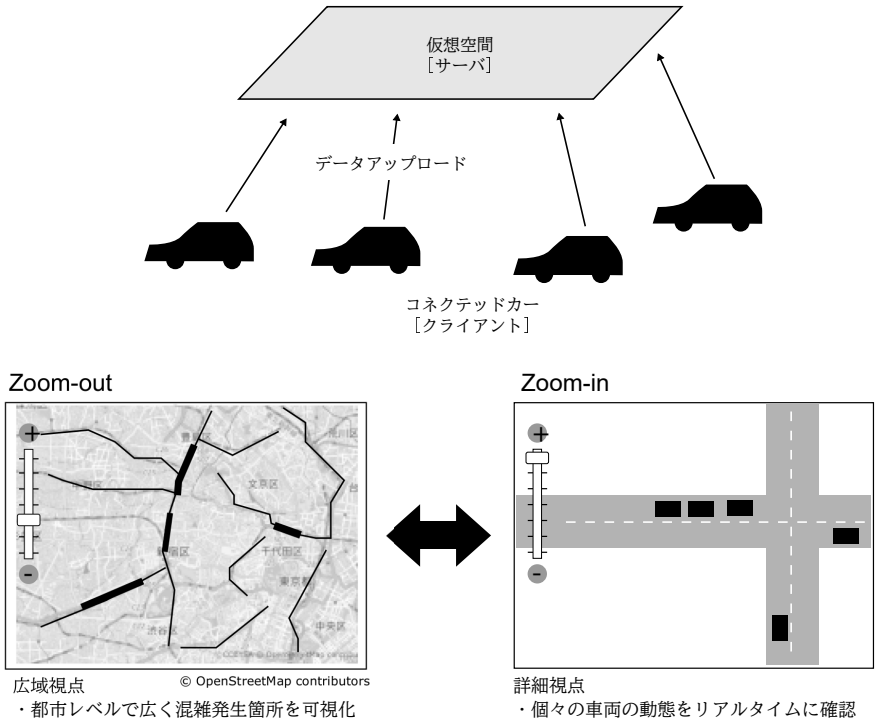


図 3.2: 交通状況を都市規模で広域に把握する基盤のイメージ

3.1.2 システムに求められる要件

前述したような使用事例を想定し，システムに対して求められる要件を述べる．

空間データ更新の内容と頻度

1.3 節で述べたように，空間データは「事前に定めた更新規則による定期更新」,「システム利用者の空間への介入による更新」,「入力された IoT データの反映による更新」のタイミングで更新されることが考えられる．このうち，最も頻繁かつ対象とする更新範囲が広いのは「事前に定めた更新規則による定期更新」である．前節で述べたサンドボックス型のゲームでは，模擬的な自然現象に沿って仮想空間を時々刻々と定期更新する必要がある．このような場合，空間格子の値を，例えば CA や偏微分方程式を差分化して得られる近接格子間の相互作用ルールにもとづいて更新することを想定している．具体的には世代 t において，仮想世界を構成する 2 次元配列の j 行，i 列における値を $c^t_{i,j}$ とすると，世代 $t+1$ における値を次の式によって求める．

$$c^{t+1}_{i,j} = f(c^t_{i-1,j}, c^t_{i+1,j}, c^t_{i,j-1}, c^t_{i,j+1}, c^t_{i,j}) \tag{3.1}$$

ここで，f は，実行するアプリケーションに依存して定まるプログラムの関数となる．また，f の引数は，更新対象の格子と，その格子の東西南北の近傍格子の値となる．多くの場合，この関数は近傍格子の状態に応じて更新対象格子の状態を決定するため IF-THEN ルールの集合や，近傍格子の値をもとに更新対象格子の値を計算するための数式によって表現されることになる．いずれにせよ，関数の内部でループを回したり，外部のデータベースにアクセスするといった，CPU に対する負荷や計算に時間のかかる処理を行うことは無い想定である．

空間の広域可視化

前章で述べたように，広大な空間を活かしたサービスを実現するには，利用者が表示縮尺を自由に変化させながら柔軟に可視化し，そこで生じている様々な空間スケールの現象を効率よく把握できることが望ましい．このためには，空間を局所的な範囲に絞って可視化するだけでなく，広い範囲を広域的に可視化できる必要がある．

また，ある時点の状況のみを可視化するのではなく，空間の変化に追従し，可視化する内容も随時更新する必要がある．空間の変化への追従性，つまり，空間データが更新された時に，それを描画反映するまでにどの程度の遅延が許容できるかについては，アプリケーションに依るところが多い．しかし，一般的な傾向として，特定の狭い範囲を詳細に可視化している時と比べて，広い範囲を俯瞰的に可視化している状況では，遅延に対する許容幅が大きいと考えている．これは，前章で述べたゲーム用途を始めとした，CA 等の近接

相互作用によって計算を行うシミュレーションでは，空間を構成する要素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にはそもそも時間がかかるためである．このため，例えばゲーム用途では，特定の範囲を詳細に可視化している時は，数秒毎に変化する空間に追従し，数秒以内にその更新内容を描画に反映する必要がある一方で，広い範囲を可視化している状況では，その縮尺（高度）に比例して，それが反映するまでに時間を要してもよい場合が多いと考えている．

クライアントの数とアクセス頻度

仮想世界にアクセスす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，ゲームとしての利用を想定した場合，格子一辺を 1m と設定した場合で 1 平方キロメートルあたり約 1,000 クライアントからのリクエストに応えられれば，多くの利用者が仮想世界を介して十分に交流することができると考えている．なお，各クライアントは時々刻々と変化する格子状況を可視化するため，視界に含まれる領域の格子データを，空間の更新頻度と揃えて数秒以内毎にリクエストすることを想定している．これらの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．

なお，クライアントからのアクセス量は，仮想世界の位置に応じて変化する可能性がある．例えばゲームの場合は，何も存在しない海エリアと，陸エリアでは，陸エリアに興味を持つプレーヤが多く，そこに対するアクセス量が海エリアよりも上回ることが予想される．

運用の形態とコスト

本章で述べるシステムは，特定の処理をバッチ的に実行するのではなく，24 時間継続動作させながら，クライアントに対してサービス提供することを想定している．そのため，基盤の運用にかかる計算機リソースやネットワークインフラの利用費用を低く抑える必要がある．また，初めは小規模な空間で運用を開始し，利用者数の増加に応じて空間の広さを漸増させていく運用形態も想定している．そのため，空間の広さを柔軟に，細かい単位で少しずつ拡大できる必要がある．

均質性と局所性

井原 [65, 122] は，自律分散システムのスケーラビリティと可用性，保守性を高めるための条件として，生物における細胞に対応したアナロジーにより均質性と局所性を挙げてい

る．均質性とはシステムの構成要素が全て同様の挙動で動作し，その機能や役割に質的な違いが無いことを意味している．均質性を有する分散システムは，システム全体の構成が単純となり，構成要素の追加や削除に伴う管理が容易になりシステムの拡張性・保守性・運用性が向上する．局所性とはシステムの構成要素が，自分や自分の近隣に位置する他の構成要素が保持する情報のみによって処理を実行できることを意味する．システムが局所性を有することで，ある地点における障害の影響が，その場所とは関係の無いところへ波及することがなく，システムの構成要素の故障時もシステム全体が停止することなく継続動作し続けることができ，システムの可用性が高まる．本章で提案する基盤はこれらのメリットを重視し，均質性と局所性を指向する．

3.2 関連研究

広域な空間データを扱う場合は，必然的に，分散して存在する複数のノードにデータを保持することになる．本節では，分散して存在するデータの可視化を対象とした具体的な技術や枠組み，およびそれらの問題点について述べる．

空間データのインデクシングを工夫する方式　Fernando [98] は，配列形式でデータを保存する ArrayDB の上でデータのインデックスを構成する方式を提唱している．これは，ラスタデータを対象に，同じ画素値を持つ塊に対して Minimum Boundary Rectangle（MBR）: 最小外接矩形を階層的に設置することでラスタデータに対する R 木インデックスを構築する手法である．このインデックスを用いて，広域表示時には下位 MBR 内のデータ取得を省略し，空間を荒く描画することで負荷を抑えることができる．しかし，データが経時変化するような状況ではインデックスをその度に再構築する必要があり，この処理に要する時間がデータの更新間隔を上回ってしまう．本章で提案するシステムは空間データを構成する多数の格子の値が頻繁に変化することを想定しており，このような状況下では，これらの空間インデクシング手法は有効に利用できない．

クエリ最適化，およびパーティショニングを工夫する方式　ArrayDB において，例えば，ある地理エリアの温度分布を表現した地理的なラスタデータを，幾つかのサブエリアに区切って，複数のノードに分担して保存させる構成を考えると，エリアを疎に区切って，地理的に近い場所にあるデータはなるべく同じノードに割当てる方式（Coarse-Grained Partitioning）と，エリアを密に細かく区切って，それぞれの小エリアをもとの地理的な距離に関係なく，なるべく分散してノードに割当てる方式（Fine-Grained Partitioning）が考えられ，それぞれ

のパーティショニング方式で，クエリに対する応答処理を実行する場合の，ノードとデータ集約部の通信量や，ノードの計算負荷の偏りが異なる．[105] では，空間データをノードへ割当てる際のデータの区切り方の粗密の度合を，あらかじめ想定したクエリ（空間データに対するユーザからの問い合わせ．例：ある領域の平均気温が知りたい）の傾向に応じて調整することで，クエリへの応答時に発生する分散ノードとデータ集約部の通信量や，ノードでの計算負荷の偏りを制御しようとするものである．しかし，本章で提案するシステムでは，多くのユーザが様々な領域を，多様な表示縮約で可視化することを想定しており，このような状況ではクエリの傾向を仮定できないため，パーティショニングのクエリ最適化を行うことができない．さらに提案システムはデータをノードに保存するだけでなく，保存したデータをノード自身がシミュレーションによって更新することを想定している．このシミュレーション実行のためには，仮想空間において地理的に近い場所のデータはなるべく同じノードが保持しておく必要があり，パーティショニングを Coarse-Grained 以外のものにすることが，そもそもできない．

分散バッチ処理して集約する方式　時空間データ可視化基盤である Map-Vis [57] は，分散バッチ処理によって並列分散処理したデータを集約部に集めて可視化する枠組みである．このシステムは，HBase の Co-Processor 機能によって分散ノード上で並列処理したデータを，集約部に集めて描画部に渡す方式が示されている．描画部については汎用の BI ツールを用いることが想定されているが，このツール用のデータ提供を行うサーバ（以降，可視化サーバ）に対する要件や性能については特に述べられていない．本章で提案するシステムでは，3.1.2 節で述べた通り，多くの利用者が空間を同時に可視化することを想定している．このような多数の利用者が可視化サーバへアクセスする場合，利用者へのデータ提供にかかる負荷を分散するため，可視化サーバを複数台動かした上で，それら全ての可視化サーバに対してデータをレプリケーションする必要がある．しかし，いったん集約したデータをレプリケーションのために再分配するのは効率が悪く，特に通信に要する時間の影響で，頻繁に変化する空間データを，変化に追従して遅延なく可視化できない可能性がある．

3.3　データのパーティショニング

3.3.1　空間データの地理分割

本章で述べる基盤では，可視化の対象とする空間を格子の集合で表現し，それを複数の計算機で並列に取り扱うため，グリッド状に分割する．本稿では以降，空間を構成する格

子 1 つ 1 つをセル（Cell），空間に含まれるセル全てを含む領域全体をワールド（World），また，分割してできた領域をサブワールド（Sub World），そして，サブワールドの一辺に含まれるセル数をサブワールド長（l_{sw}: Length of Sub World），ワールドの一辺に含まれるセルの数をワールド長（l_w: Length of World）と呼ぶ．そして，サブワールドの中に含まれるセルの数をサブワールド面積（a_{sw}: Area of Sub World），ワールドの中に含まれるセルの数をワールド面積（a_w: Area of World）と呼ぶ．また，サブワールドのセルを行列で表現した時，i 列 j 行目に位置するセルを $c_{i,j}$ と表記する．これらを図 3.3 に示す．

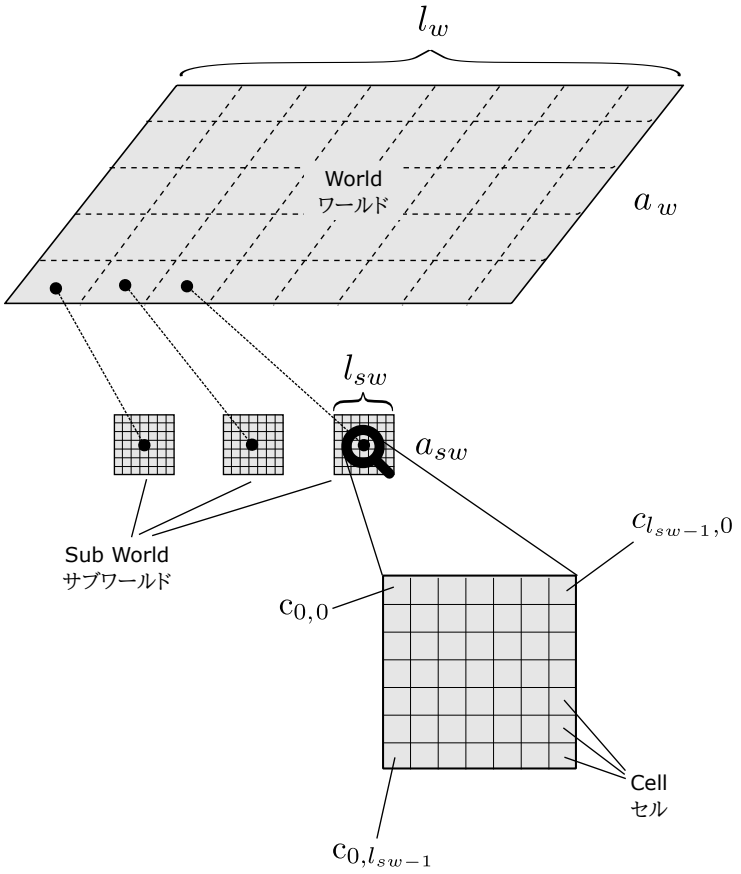


図 3.3: 空間データの地理分割と本論文内での用語

3.3.2 計算ノードの格子状ネットワーク接続

提案方式では，複数の計算機をネットワークで格子状に仮想的に配置し，ワールドを分割したサブワールドを 1 つを 1 つの計算機に割り当てる（図 3.4）．本稿では以降，この計算機をノード（Node）と呼ぶ．各ノードは自らに東西南北で近接する 4 つのノードと接続し，データの送受信を行う．

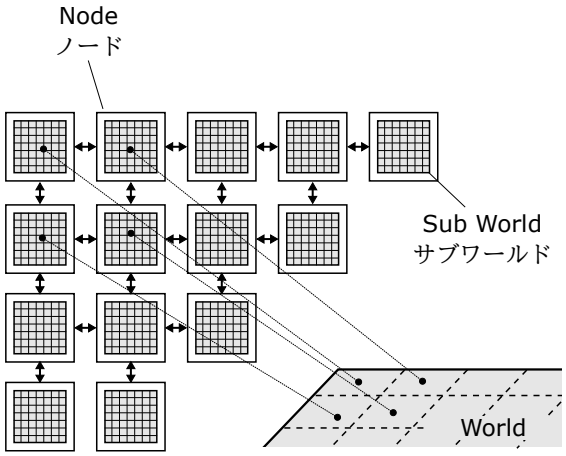


図 3.4: ノードの格子状ネットワーク接続

空間上で時間発展シミュレーションを行うような場合，ノードは，自らが保持するサブワールドを構成するセルを式（3.1）によって定期更新する．更新のタイミングは後述するモード（同期，非同期）によって異なる．時間発展シミュレーションを行う場合，サブワールドには世代があり，セルの更新を行う度に世代数が 1 つ加算される．本稿では以降，ノードが保持するサブワールドの世代を「ノードの世代」と呼ぶ．

3.3.3 境界セルの更新処理

セルの更新時，ワールドの分割境界に位置するサブワールドの位置する境界セル $c_{0,j}, c_{l_{sw-1},j}, c_{i,0}, c_{i_{sw-1},0} \mid i, j \in 0, 1, \cdots, l_{sw-1}$ については，その値を更新するためには近接する他のノードが保持するセルの情報が必要となる（図 3.5）．

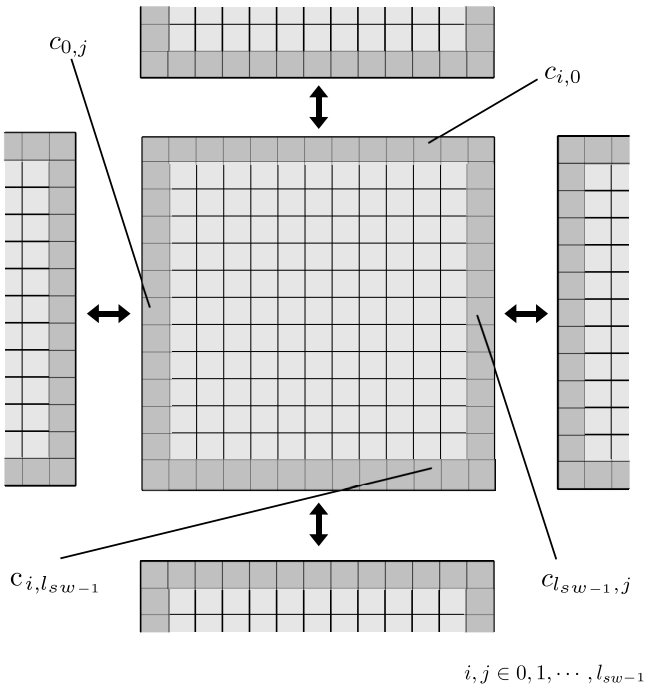


図 3.5: 境界に位置するセルの更新

本基盤では，後に述べるデータのレプリケーション処理によって，近接するノードから送信されてくるデータの中に，これら境界セルの更新に必要な情報が含まれるため，それを用いて更新を行う．

3.3.4 同期モードと非同期モード

空間の時間発展シミュレーションを行う場合，前節で述べたように，ノードが自らが保持するサブワールドの境界セルの値を更新するためには，近接ノードが保持するデータが必要である．これらのデータの受信と計算実行タイミングには同期モードと非同期モードが考えられる．

同期モード

セルの更新処理を行うたびに，全ての近接ノードからデータを受信するまで待機する（バリア同期に入る）．本基盤ではノードが格子状にネットワークで接続されており，個々のノードが近隣ノードと同期をとって動作すれば，結果として全てのノードが同期して世代が揃う．このため，正確なシミュレーションを実行することができる（図 3.6）．しかし，ノードにおける一時的な処理遅延や，ネットワーク輻輳遅延の影響によって発生した同期の

遅れは連鎖的に他のノードへ波及し，原理的には他全てのノードが影響を受ける（図 3.7）．大量のノードが稼働することを想定した場合，一時的な同期に遅れが生じるノードは常に存在し，それら全ての影響が全体に及ぶため，結果的に全体のシミュレーション実行速度は著しく低下する．このことから，ノード数が多い場合は完全な同期モードでの実行は実質難しく，基本的には同期モードで稼働しつつ，同期待ちの時間が閾値を超えた時点で一時的に後に述べる非同期モードで動作するといった対策が必要となる．

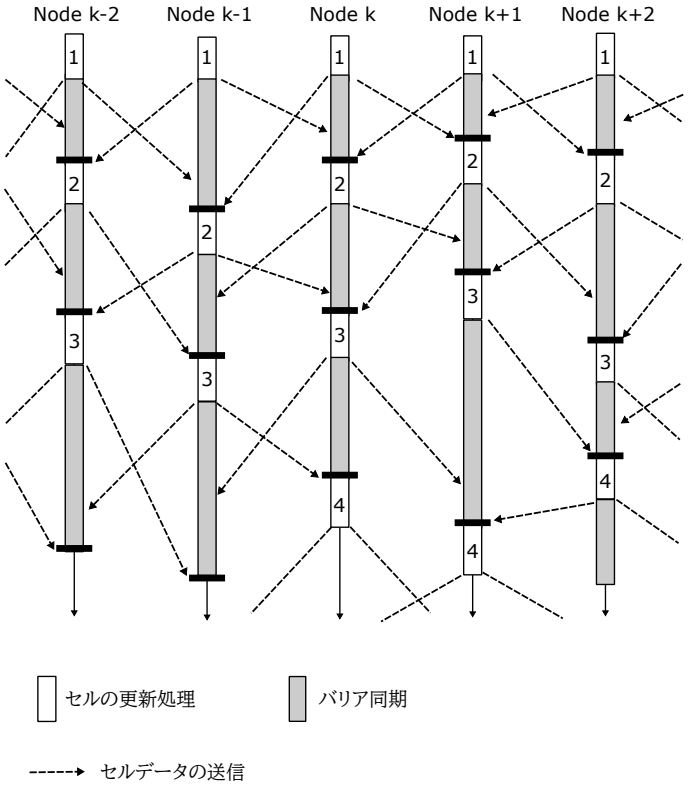


図 3.6: 同期モードの動作シーケンス．数字はセルの世代を示す．簡単のためノードを 1 次元で接続した構成で図示する．同期モードでは左右両方の近隣ノードからデータを受信するまで待機（バリア同期）する．

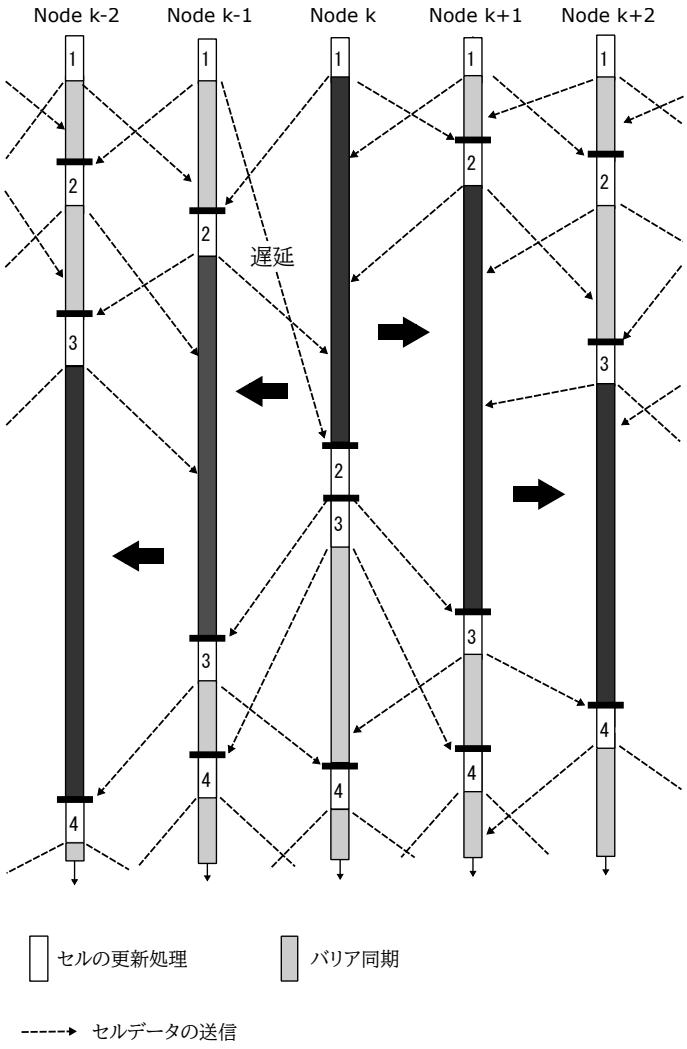


図 3.7: 同期モードで生じる遅延の連鎖波及．ここでは Node k-1 から Node k に対するデータ送信に遅延が発生し，それが周辺のノードへ波及する様子を示している．

非同期モード

各ノードが常に一定の間隔でセルの更新処理を実行する．近接ノードのデータ受信を待つことなく計算を行うため，同期モードで生じるような全体遅延は発生しない．しかし，各ノードは近接ノードと同期を行わないため，ノードの世代はシステム全体で揃わなくなる．全てのノードは共通の時間間隔で更新処理を実行するが，その時間を計測する時計は共通ではなく，ノードが各自のホストの時計を用いる．また，計算処理に要する時間はその時々のノードの負荷によって若干異なるため，こうした誤差が積算することで，計算世代の進行にノード毎の差異が生じ，シミュレーションの進行がサブワールド毎に微妙に異なることになる．非同期モードでのノードの動作シーケンスを図 3.8 に示す．

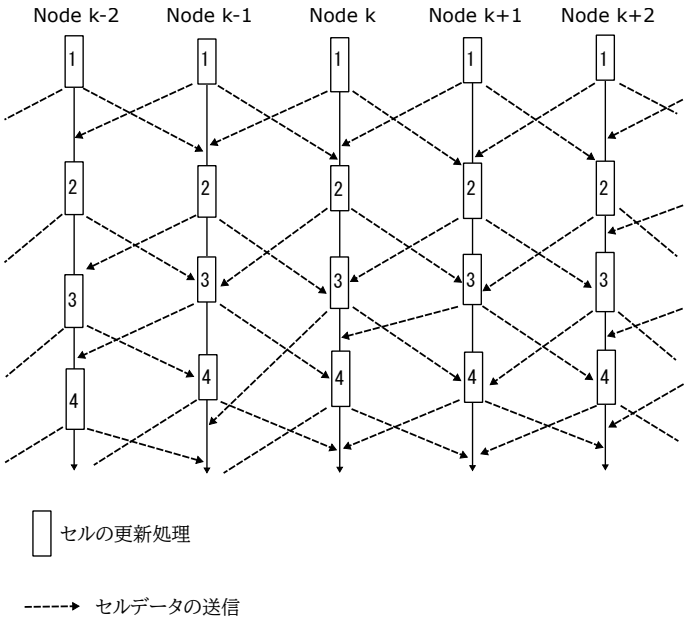


図 3.8: 非同期モードの動作シーケンス．数字はセルの世代を示す．簡単のためノードを 1 次元で接続された構成で図示する．非同期モードでは，近接ノードからのデータ受信を待つことなくセルの更新処理を定期的に実行する．

一般的に分散システムの同期は大きな課題であり，これを効率化するための方式も研究されている（ [119, 124]）．しかし，同期を行う限りは処理の遅延を原理的に免れない．特に本基盤の場合は，同期を行う場合は，数秒以内の間隔で定期的に生じるセル更新の度にバリア同期に伴う待機が発生するため，遅延が頻繁に発生することになる．著しい遅延発生は，本基盤が重視するインタラクティブ性を損ねる原因となる．

一方で，非同期の場合や，同期を緩和できる場合，遅延の問題は大きく解消する．同期の緩和とは，セル更新の度に同期を行うのではなく間欠的に同期を行ったり，上述したように遅延発生状況に応じて適応的に同期を抑制したりといった対処を行うことである．精度が要求される科学シミュレーションなどでは，このような同期の廃止や緩和を行うのは精度要件から難しく，多くの場合，完全な同期は実施される．しかし，本基盤の想定する利用用途は 3.1 で述べたように，インタラクティブなゲームや交通流シミュレーションなどである．これらは計算の厳密性や精度よりも，ゲームとして違和感のなさや，人や車両の流れの定性的な挙動の再現といったことが求められる．したがって，本基盤は主に非同期モードでの動作を想定している．非同期モードによってアプリケーション側に生じる制約に対する考えは，3.7 で述べる．

3.4 データのレプリケーション

本提案では，利用者が仮想空間を可視化するために必要となるデータを，ノード間で複製する．この目的を次に示す．

広範囲なデータを遅延なく利用者へ提供するため

各ノードが，広域表示に必要となる広範囲のサブワールドデータを，周囲のノードから複製して保持しておくことで，広範囲な可視化要求に対して，必要となるデータをその都度収集することなく迅速に利用者へ提供する．

クライアントからのアクセス負荷分散のため

広域表示用のサブワールドデータを複数のノードが重複して保持することで，クライアントからのデータ取得要求をノード間で分散することができ，単一ノードにかかる負荷を抑制する．

3.4.1 データの階層化と拡散

2.1.4 で述べたように，広域な空間データを可視化するためには，元のデータから段階的に LOD を落としたデータピラミッドを生成しておき，クライアントが求める表示縮尺に応じて異なる LOD のデータを提供することが効率的である．本基盤では，ノードは自らが保持するサブワールドのデータからデータピラミッドを生成し，サブワールドのセルを更新するたびに，データピラミッドも更新する．

図 3.9 は，ノードが自ら生成するサブワールドのセルデータから，再帰的に解像度を下げて生成するデータセットを示したものである．本稿ではサブワールド長 l_{sw} を 2^s と置いたときに，そのデータの LOD を s とし，そこから $s-1, s-2, \cdots$ と LOD を下げたデータを生成する．したがって，l_{sw} が定まればその系の最大の LOD が定まり，これを最大 LOD（d_{max}）と呼ぶ．また，解像度を下げる回数を LOD 階数（dd_n）と呼ぶ．そして，LOD の下限を最小 LOD（d_{min}）と呼ぶ．これらの関係をまとめると次の通りになる．

$$\begin{aligned} d_{max} &= s, \\ d_{min} &= d_{max} - dd_n \\ l_{sw} &= 2^{d_{max}} \end{aligned} \tag{3.2}$$

本稿では以降，必要に応じて解像度が落とされたデータを，そのデータの LOD を含めて ”サブワールド（LOD）” という形式で記載する．LOD と，それに対応するサブワール

ドのサイズを表 3.1 に示す.

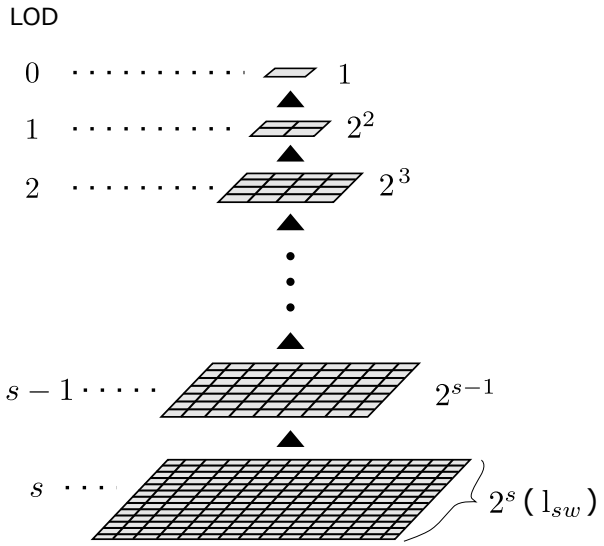


図 3.9: サブワールドの LOD

解像度を落とす方法はアプリケーションに依って異なるが，大きく分けて，代表値選択方式と，平均化方式がある（図 3.10）．代表値選択方式はサンプリング対象のセル群から，解像度を落とした時のセルの値を選択的に決定する方式である．例えば 9 つのセルを対象としてサンプリングする際に，そのセル内の中央に位置するセルの値や，あるいは最も出現頻度の多い値を，解像度を落とした際のセルの値として採用する．これに対して平均化方式は，サンプリングするセルの数値的な平均値を採用する．平均化方式の場合はセルが離散的な属性値ではなく，連続的な量を持つ値である必要がある．

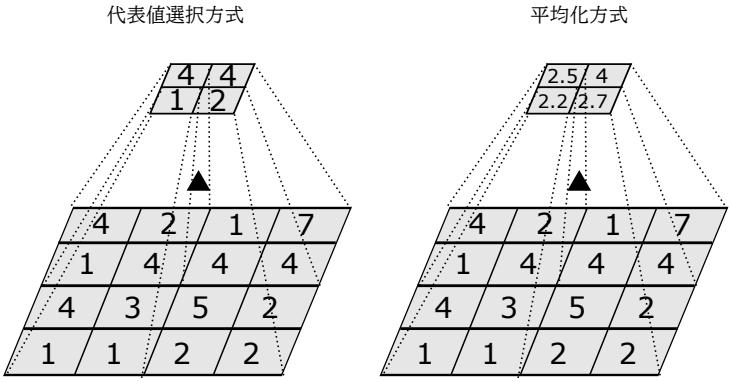


図 3.10: 解像度を落とす方式

LOD	l_{sw}：サブワールド長	a_{sw}：サブワールド面積	データサイズ
0	2^0	2^0	2byte
1	2^1	2^2	8byte
2	2^2	2^4	32byte
3	2^3	2^6	128byte
4	2^4	2^8	512byte
⋮	⋮	⋮	⋮
10	2^{10}	2^{12}	16.384Kbyte

表 3.1: LOD とサブワールドのサイズ（セル 1 つあたりのデータサイズを 2 バイトとして計算）

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，自らが保持するサブワールド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サブワールド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，全てのノードが，そのノードの周辺に存在するノードのサブワールド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サブワールドデータを自らが生成したサブワールドデータと併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．

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 [97]．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上で，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

像度が荒く LOD の低い広域表示用のサブワールドデータ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サブワールドデータは，限定されたノードに局所的に存在していればよい．図 3.11 に，このような分布の例を模式的に示す．図 3.11 において，LOD が s のサブワールドデータは各ノードが 1 つずつ保持している．LOD が $s-1$ のサブワールドデータは，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 $s-2$ のサブワールドデータは，距離 2 以内の近傍ノードとサブワールドデータを相互に重複保持している．

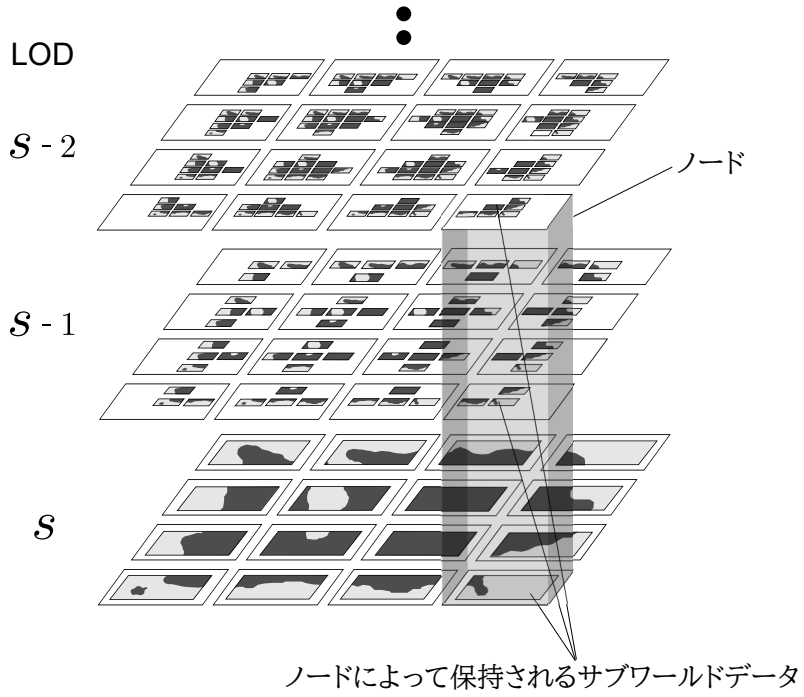


図 3.11: 負荷分散のための理想的な空間データ分布の概念図

このような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サブワールドデータを配信する．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サブワールドデータと併せて伝播する．サブワールドデータには後述する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．これを全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サブワールドデータは，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．この過程を図 3.12 に模式的に示す．

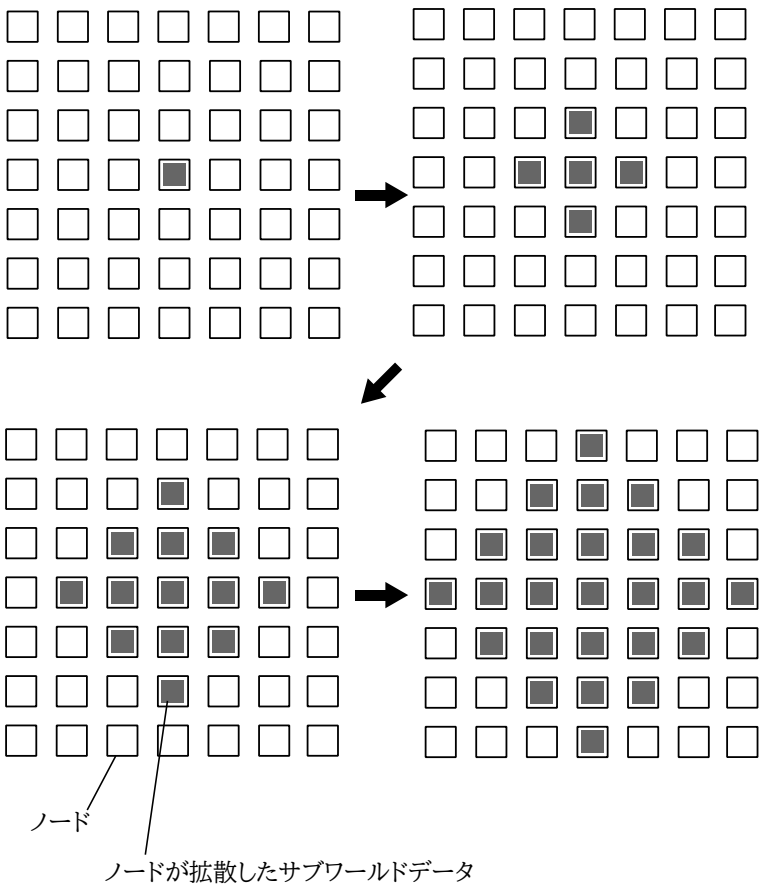


図 3.12: ノード間のデータ拡散過程

3.4.2 LOD 切替セル数と限界視点高度

提案方式で扱う仮想空間は２次元であり，これを可視化する際のユーザの視点カメラの位置は３次元直交座標系上の点として表現する．また，カメラの向きは，仮想空間に対して垂直固定とし，視野角は水平，垂直ともに 90 度とする（図 3.13）.

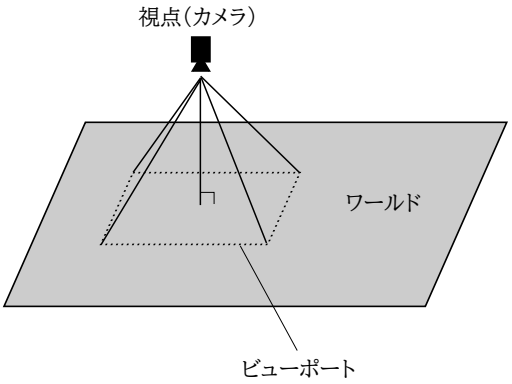


図 3.13: カメラとビューポート

仮にセル一辺の長さを 1 メートル，高度を h とすると，カメラの高度と，その時にビューポートに含まれるセルの数は $(2h)^2$ となり，高度が上昇するに連れて表示する必要のあるセル数は指数関数的に増加してセルデータの取得やその描画にかかる負荷が増大する．ここで，LOD 切替セル数（a_{swt}）とは，ビューポートに含むセル数の限界を定めるものであり，カメラの高度上昇に伴ってビューポート内のセル数が a_{swt} に到達するたびに，表示する LOD を 1 段階下げる．LOD を 1 段階下げると，その時点で表示するセル数は $\sqrt{a_{swt}}$ になるため，負荷を軽減することができる．

なお，表示する LOD が最小 LOD（d_{min}）である場合は，表示セル数が a_{swt} に到達しても，それ以上の高度上昇は行わないものとする．つまり，a_{swt} と d_{min} によって，カメラの高度の上限が規定されることになる．カメラの高度上昇に伴って LOD を切り替える過程を図 3.14 に模式的に示す．また，この時の高度と表示セル数の関係を図 3.15 に示す．

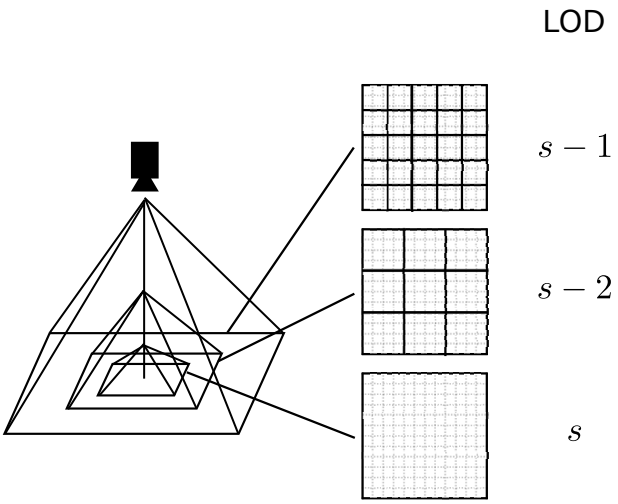


図 3.14: 視点高度上昇に伴う LOD の切替

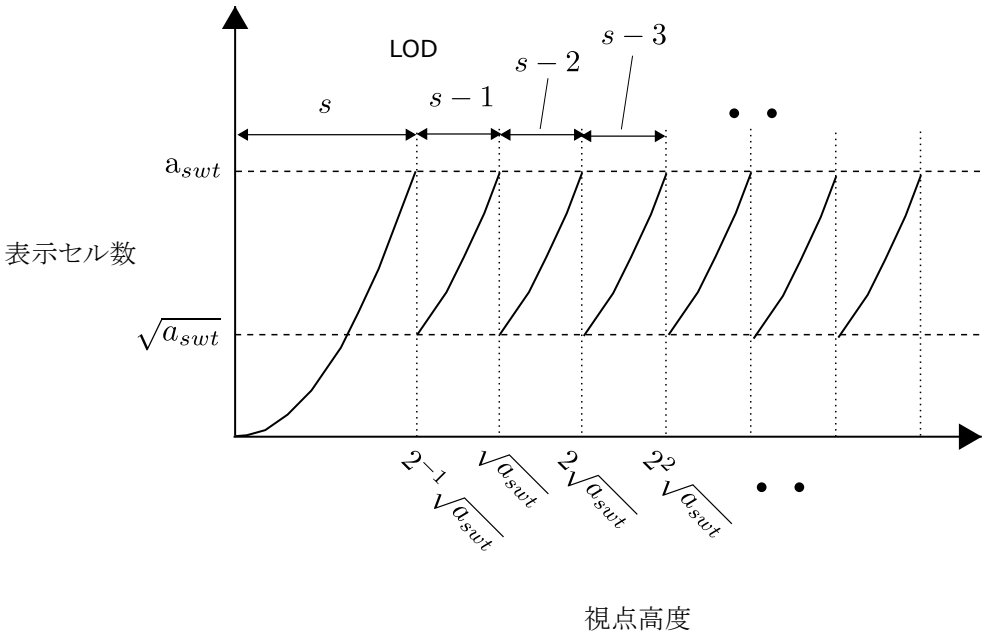


図 3.15: 視点高度と表示セル数の関係

3.4.3 拡散半径の決定方法

図 3.16 に，視点高度の上昇に伴った LOD 切替のタイミングとその時のビューポートの一辺の長さ（セル数：l_{vp}）を示す.

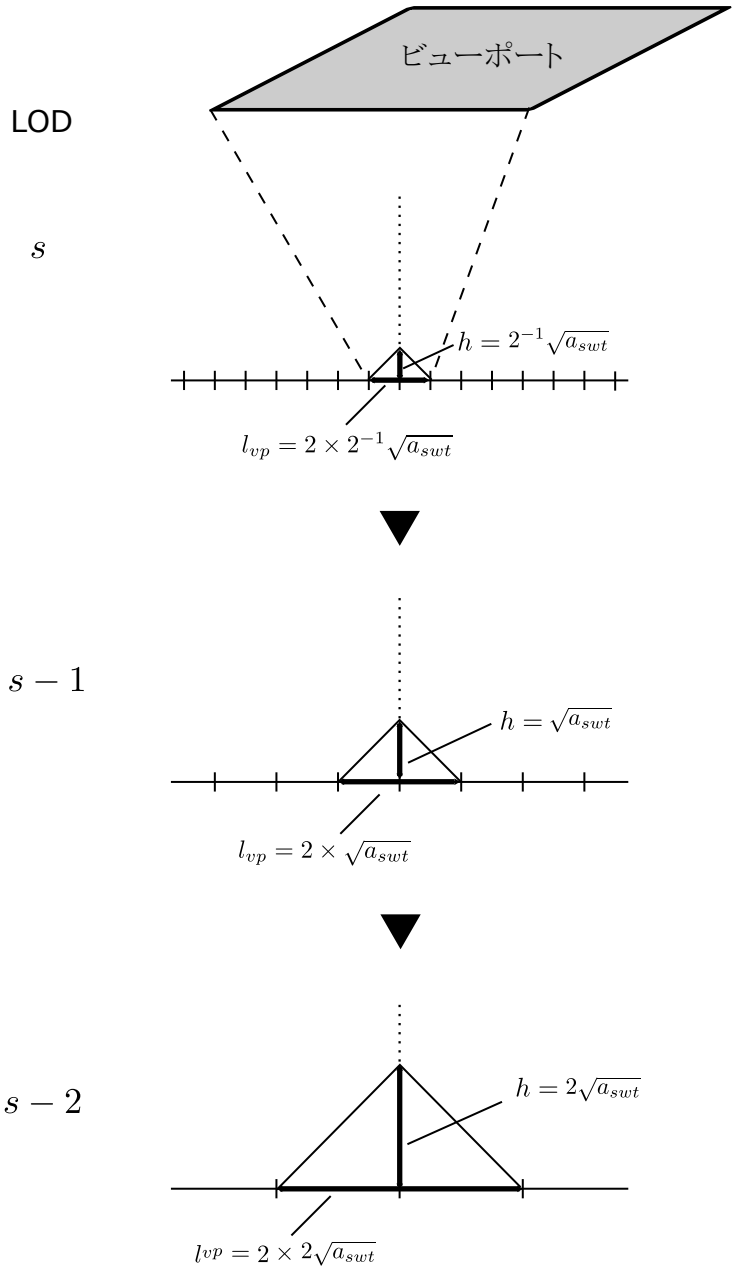


図 3.16: 視点高度とビューポートの一辺長 (l_{vp})

図 3.15 から，高度を 0 から上昇させていった際に k 回目に LOD が切り替わる視点高度（h_k）は次の式の通りとなる．

$$\begin{aligned} h_k &= 2^{k-1}\left(\frac{1}{2}\sqrt{a_{swt}}\right) \quad \{k \mid k \in \mathbb{Z}\} \\ &= 2^{k-2}\sqrt{a_{swt}} \end{aligned} \tag{3.3}$$

また，図 3.16 より，視点の高度が h_k のとき，その視点における l_{vp} は次の式によって記述できる．

$$l_{vp} = 2h_k \tag{3.4}$$

さらに，l_{vp} の中に，サブワールドの一辺が含まれる個数（n_{vps}）は，

$$\alpha = \frac{l_{vp}}{l_{sw}} \tag{3.5}$$

と置いて，

$$\begin{aligned} \lfloor \alpha \rfloor \leq n_{vps} \leq \lfloor \alpha \rfloor + 1 \quad (\alpha \geq 1) \\ n_{vps} = 1 \quad (\alpha < 1) \end{aligned} \tag{3.6}$$

となる．なお，式（3.6）における不等式は，図 3.17 のように，ビューポートの位置に応じて，その一辺が含む領域のデータを保持するノードの数が異なることに由来する（図 3.17）.

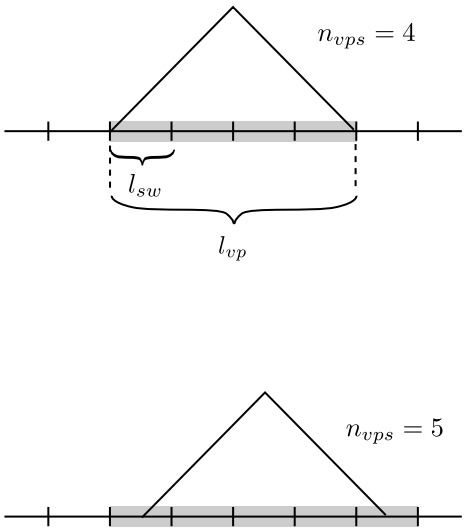


図 3.17: ビューポート位置に応じた n_{vps} の違い

式 3.6 によって表されるビューポート矩形の辺に含まれるノードの数をそれぞれ n^1_{vps},n^2_{vps} と置くと，ビューポートが被覆するノードの数（ビューポート被覆ノード数：m_{vps}）は，

$$m_{vps} = n^1_{vps} \times n^2_{vps} \tag{3.7}$$

となるため，

$$\begin{aligned} \lfloor \alpha \rfloor^2 \leq m_{vps} \leq (\lfloor \alpha \rfloor + 1)^2 \quad (\alpha \geq 1) \\ m_{vps} = 1 \quad (\alpha < 1) \end{aligned} \tag{3.8}$$

となる．例えば $\lfloor \alpha \rfloor = 4$ の時，式 3.8 によると m_{vps} は $4 \leq m_{vps} \leq 9$ となる．m_{vps} が 4 から 9 までそれぞれの値を取るときの状況を図 3.18 に示す．

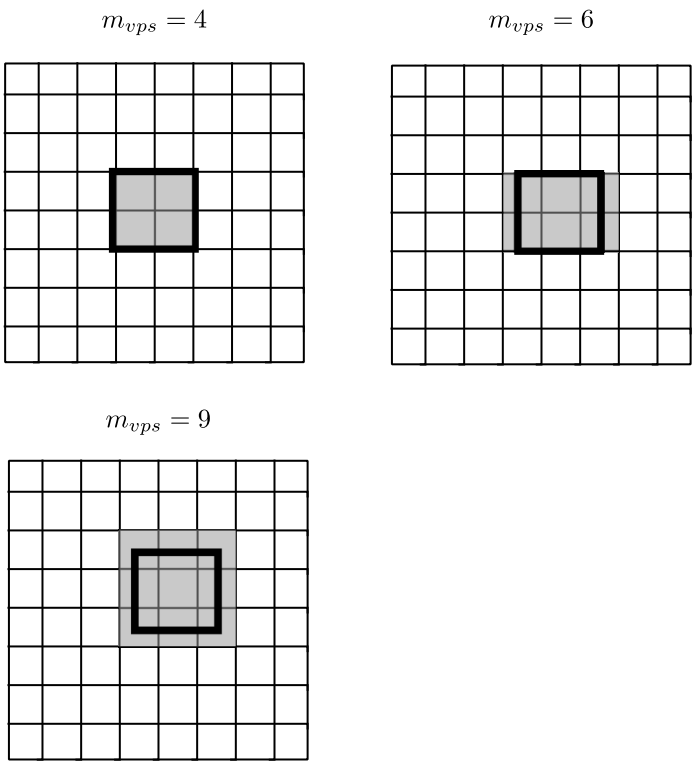


図 3.18: ビューポート位置に応じた被覆ノード数の違い

式（3.6），および式（3.5）より，LOD が $d_{max} - k$ のとき，ビューポートの一辺に含まれるサブワールドの個数は，高々次の式の通りとなる．

$$\begin{aligned} n_{vps} &= \left\lfloor \frac{2^{k-1}\sqrt{a_{swt}}}{2^{d_{max}}} \right\rfloor + 1 \\ &= \left\lfloor 2^{k-1-d_{max}}\sqrt{a_{swt}} \right\rfloor + 1 \end{aligned} \tag{3.9}$$

さらに，ここで LOD 切替セル数（a_{swt}）を 2^{2s}，LOD を K（$K = d_{max} - k$）と置くと，式（3.9）は

$$n_{vps} = \left\lfloor 2^{s-K-1} \right\rfloor + 1 \tag{3.10}$$

と書ける．

3.3 節で述べた通り，本基盤では格子状にネットワークで接続したノードがそれぞれ一つのサブワールドデータを保持する．そのため式（3.10）で示した n_{vps} は，図 3.19 のように，

ビューポートに含まれる範囲のデータを保持しているノード群によって形作られる正方形の一辺に含まれるノード数を表しているとも言える．

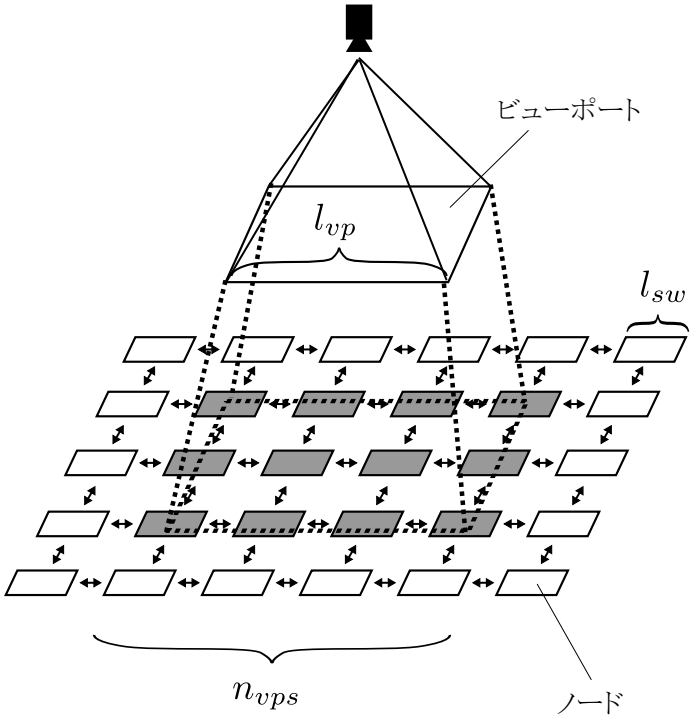


図 3.19: ビューポートに含まれるノード群と n_{vps}

また，3.4 節で述べた通り，クライントはノードにアクセスすることで可視化用のデータを取得する．クライアントが，現在のビューポート範囲のサブワールドデータを可視化するためにアクセスするノードは，そのビューポートの中央に位置するノードである．ビューポートの位置はユーザが任意に変更可能であるため，全てのノードは，そのノードを中心としたビューポート範囲内のサブワールドデータを保持しておく必要がある．また，ビューポートの範囲は LOD によって変化するため，LOD 毎のサブワールドデータごとに異なる範囲内のノードが生成するデータを保持しておく必要がある．

前述したように，本基盤は，各ノードがバケツリレー方式でサブワールドデータを周囲のノードに伝播させる．このため，２つのノード間でデータを伝達するためにかかる伝播回数（ホップ数）は２ノード間のマンハッタン距離と等しくなる．ビューポートの中央に位置するノードと，そこから周辺のノードとのマンハッタン距離を図 3.20 に示す．

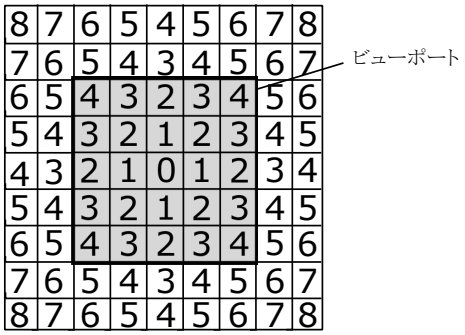


図 3.20: ビューポートの中央に位置するノードと，そこからのマンハッタン距離

図 3.20 において，ビューポートの中央に位置するノードと最も遠い場所に位置するノードはビューポートの四隅に存在し，このビューポートの例ではそのマンハッタン距離は 4 である．また，ビューポートの一辺に含まれるノード間のネットワーク接続リンク数も 4 となる．一般的に，座標平面上の二点 $A(a_1, a_2), B(b_1, b_2)$ 間のマンハッタン距離 $d(A, B)$ は

$$d(A, B) = |a_1 - b_1| + |a_2 - b_2| \tag{3.11}$$

となる．ここでビューポート一辺のノード間リンク数を s とし，ビューポートの中央に位置するノードを A，そこからビューポート内の最遠ノード（四隅のノード）を B とおくと，ビューポートは正方形であるため $|a_1 - b_1|$ も $|a_2 - b_2|$ もともに $s/2$ となり，結果的に $d(A, B) = L$ となる．すなわち，ビューポートの中央に位置するノードと，そこからビューポート内の最遠ノードとの距離は，ビューポートの一辺に含まれるノード間の接続リンク数と等しく，それは $n_{vps} - 1$ となる．したがって，そのビューポートに含まれるノードが，全て $n_{vps} - 1$ の距離だけサブワールドデータを周辺のノードに拡散すれば，ビューポートがどこに位置していても，そのビューポートの中央に位置するノードが，そのビューポートに含まれる全ノードのサブワールドデータを保持することになる．

以上を踏まえ，LOD 切替セル数（a_{swt}）を 2^{2s}，LOD を K（$K = d_{max} - k$）とした時，サブワールドデータ（K）の拡散半径 r_K を，

$$r_K = \begin{cases} 1 & (s < K - 1) \\ 2^{s-K-1} & (S \geq K - 1) \end{cases} \tag{3.12}$$

によって定める．この式と，高度と LOD の関係の式（3.3）から，高度に応じた LOD と視野内ノード数を，$d_{max} = 10, a_{swt} = 2^{12}$ の場合を例にとり表（3.2）に示す．また，図 3.21 に，$d_{max} = 9, a_{swt} = 2^{12}$ の時のデータ分布状況を模式的に空間一次元で示す．

LOD	h_k：LOD 切替高度	n_{vps}：ビューポート被覆ノード数（一辺）	r_K：拡散半径
0	$h_{10} = 2^{14}$	$2^7 + 1$	$r_0 = 2^7$
1	$h_9 = 2^{13}$	$2^6 + 1$	$r_1 = 2^6$
2	$h_8 - 2^{12}$	$2^5 + 1$	$r_2 = 2^5$
3	$h_7 = 2^{11}$	$2^4 + 1$	$r_3 = 2^4$
4	$h_6 = 2^{10}$	$2^3 + 1$	$r_4 = 2^3$
5	$h_5 = 2^9$	$2^2 + 1$	$r_5 = 2^2$
6	$h_4 = 2^8$	$2^1 + 1$	$r_6 = 2$
7	$h_3 = 2^7$	2	$r_7 = 1$
8	$h_2 = 2^6$	1	$r_8 = 1$
9	$h_1 = 2^5$	1	$r_9 = 1$
10	0	1	$r_{10} = 1$

表 3.2: 視点高度と視野内ノード数，LOD の関係（$d_{max} = 10, a_{swt} = 64^2$ の場合）

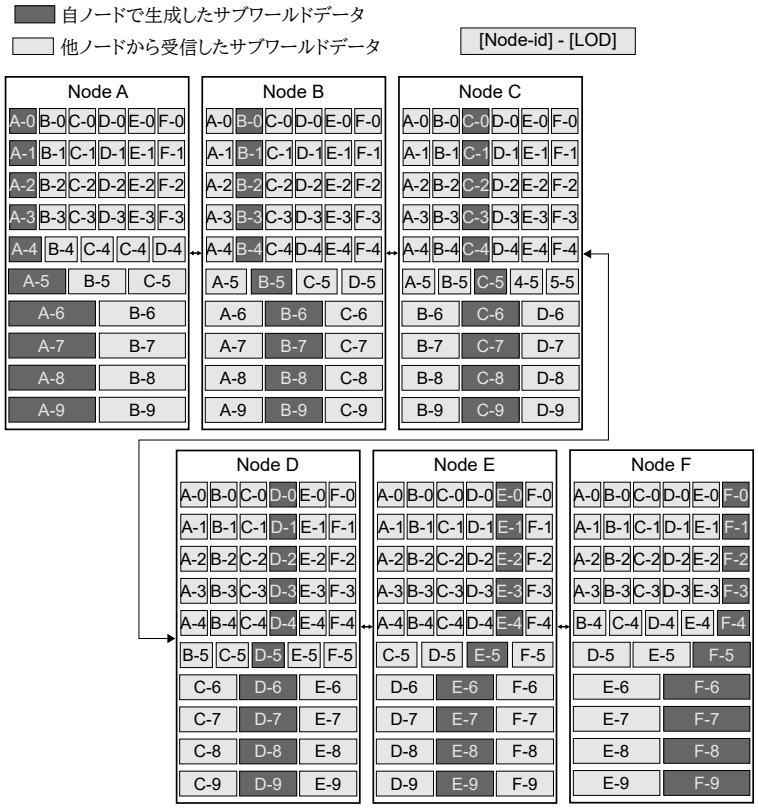


図 3.21: $d_{max} = 9, a_{swt} = 2^{12}$ の時のサブワールドデータ分布の例．説明のため，ノードが空間 1 次元で接続された単純な構成で示した．

3.5 実装

3.5.1 システム構成と動作方式

前章で述べた方式によって動作するノードを Java によって実装した．ノードは図 3.22 に示す通り，1 つの仮想マシン（VM）につき，1 つのプロセスを常駐稼働させることを想定している．ノード間通信は Socket（TCP）によって行い，送受信するデータは ZIP で圧縮・解凍する．

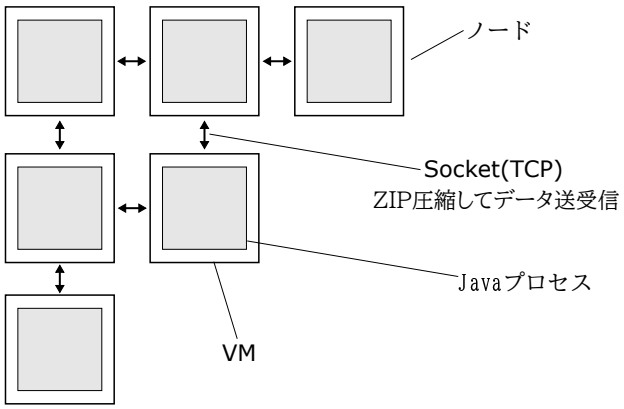


図 3.22: ノードの実装形態

なお，今回の実装では，ノードのプロセスを起動する際に，ノードに対してそのノードの近傍ノードの IP アドレスを設定値として与えるようにした．実際には，ネットワークに参加するノードは付録 A.2 で示す方式により近接ノードのアドレスを入手する．

3.5.2 ノードの構成と機能

ノードの構成を図 3.23 に示す．

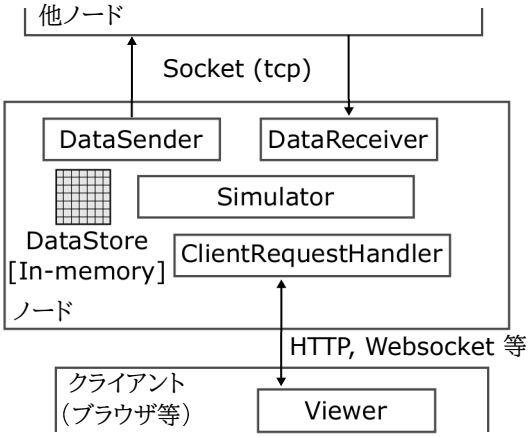


図 3.23: ノードの構成

図 3.23 で示したクラスのうち，主要なクラスの役割を下に示す．

Node

ノードのメインクラス．

DataStore

自らが生成するデータや，他のノードから送信されてきたデータをメモリ上に保持管理するクラス．

DataSender

サブワールドデータを含んだメッセージを定期的に ZIP 圧縮して東西南北の近隣ノードに対して送信するクラス．

Simulator

サブワールドデータの時間発展をシミュレーションするクラス．具体的には自らが生成するサブワールドデータ（d_{max}）の配列をラスタ的に走査し，データ内の各セルを，その周辺セルの状態値にもとづいて更新する処理を定期的に実行する．

ClientRequestHandler

ブラウザなどからアクセスしてきたクライアントに対してサブワールドデータの可視化や編集を行う Web クライアントプログラムを提供するサーバ．またそのクライアントプログラムからの要求に応じてデータの提供や更新を行う．

ノードの動作フロー

図 3.24 にノードの動作フローを示す.

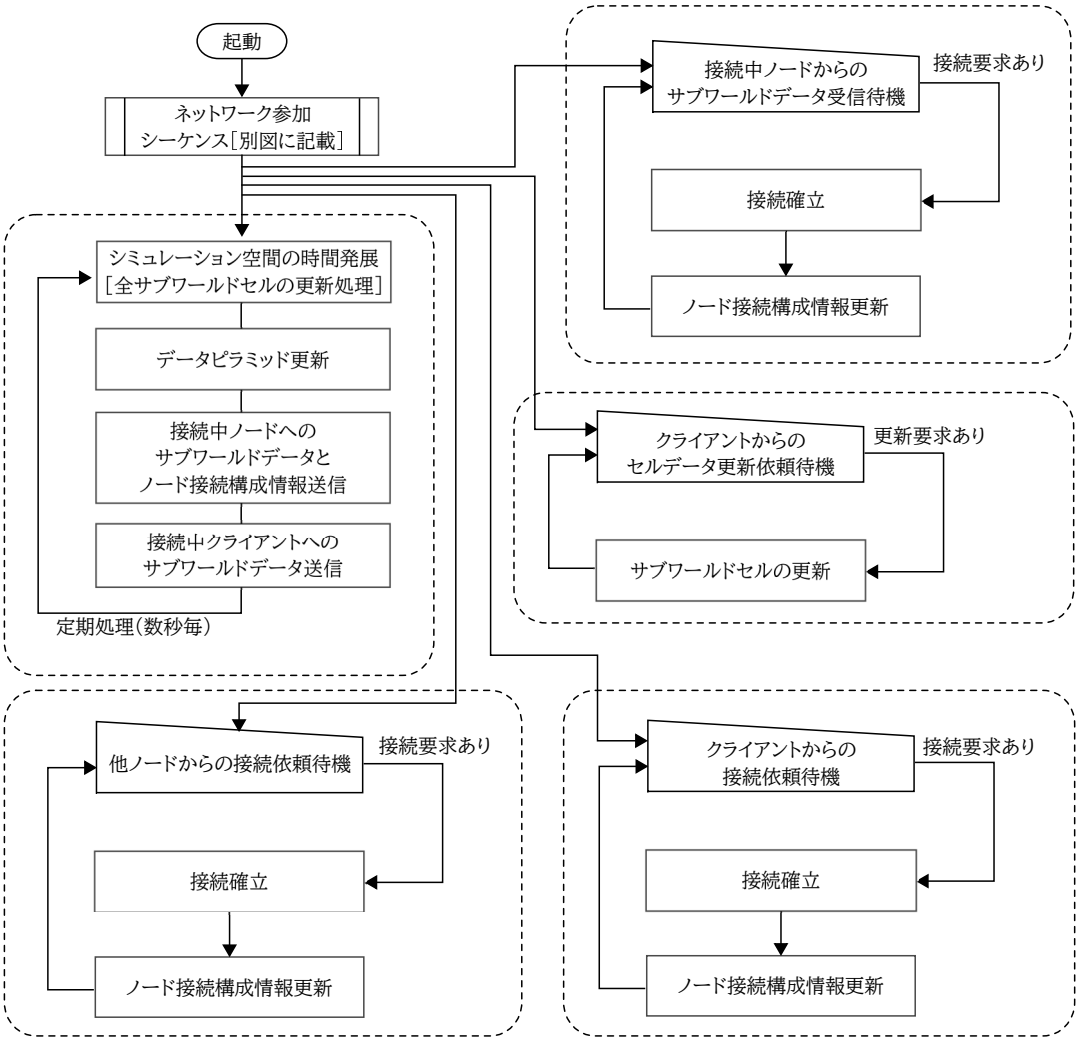


図 3.24: ノードのフロー

3.5.3 クライアントの構成

本基盤におけるクライアントは，次のいずれか，あるいは両方の機能を持つ.

1. ノードからサブワールドデータを継続的に受信し，可視化してユーザに提示する.

2. ノードに対してサブワールドを構成するセルの更新依頼を送信する.

クライアントの形態や詳細な役割はアプリケーションに応じて異なる．また，アプリケーションに依っては異なる形態のクライアントが存在する可能性もある．例えば 3.1 節で述

べた交通シミュレーションの例では，車両の走行データをアップロードし，セルの更新を依頼するクライアントと，交通状況を可視化するためにサブワールドデータを受信するクライアントが別々に存在することも考えられる．

この節では，主に 3.1 節で述べたゲームを想定したクライアントとして，ユーザが PC にクライアントアプリケーションをインストールすることなく，サブワールドの可視化や編集をブラウザから行えるようにするための構成を示す．この構成では，ノードは HTML/Java スクリプトで動作するブラウザアプリケーションをクライアントへ提供する機能を持つ．ブラウザアプリケーションは，対ユーザ向け UI や，WebSocket による継続的な通信機能を持つ．クライアントとサーバの関係を図 3.25 に示す．

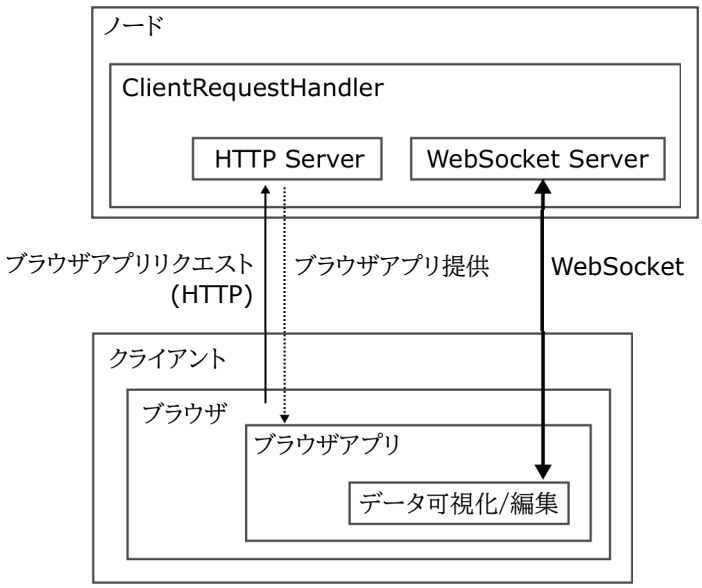


図 3.25: ノードとブラウザクライアントの関係

また，クライアント側で空間を可視化するまでの流れを例に，ノードとブラウザクライアント間の通信シーケンスを図 3.26 に示す．

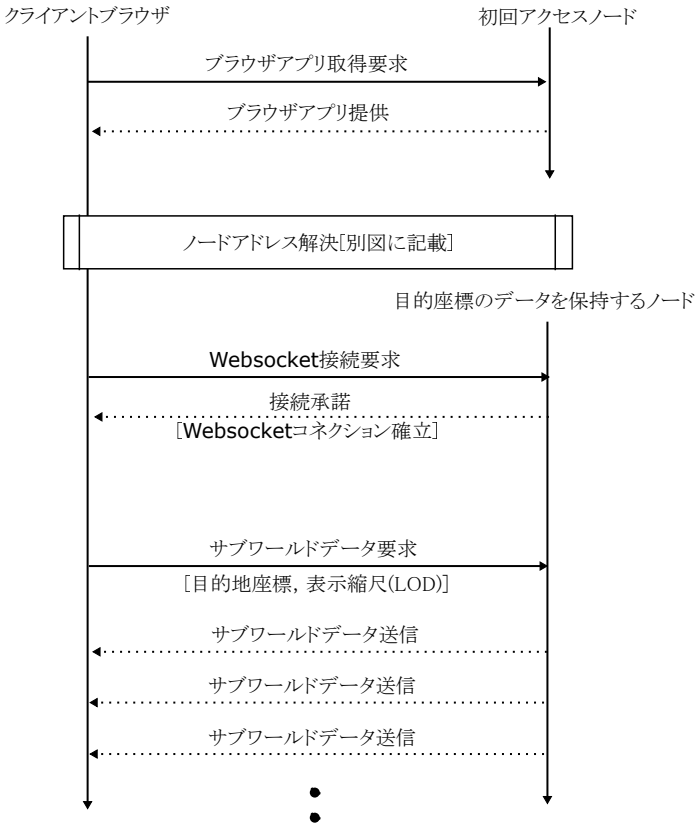


図 3.26: ノードとブラウザクライアントの通信シーケンスの例

3.6 評価実験

3.6.1 評価の概要

前節で述べた方式によって稼働するシステムを対象として，ノードの台数やユーザ数の増加に対する性能を検証することを目的として，次の評価を実施した.

（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伴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仮想空間の広さを拡大した時のスケーラビリティを検証する.

（評価２）クライアントからのリクエスト数増加に伴う負荷変動（従来方式との比較）

クライアントからのリクエスト数の増加に伴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上で，実験により定量的な差を示す.

（評価３）クライアントからのリクエスト数増加に伴うレスポンス時間の変動

クライアント数の増加に伴う，ノードからクライアントへのレスポンス時間の増加

傾向を確認し，遅延が大幅に増加しないことを示す.

（評価４）ノード間のデータ流通に要する時間

ノードが生成するデータがマルチホップで他のノードに到着するのに要する時間を計測し，データレプリケーションの遅延の程度を確認する．また，ノードがデータセンタ内の同一の場所に存在する場合とそうでない場合の遅延の傾向の差を確認する.

評価環境・条件

評価にあたり，ノードは AWS EC2 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（Region: データセンターが存在する物理的なロケーション），同一アベイラビリティゾーン（Availability Zone[AZ]: リージョン内に存在する，電源やネットワーク設備の冗長化単位）の中に配備して分散システムを構成した．ただし評価４ではノードを異なる AZ に配置する構成もとっている．インスタンスの OS には CentOS 7 を用いた．計測の対象とした基盤の構成を図 3.27 に示す.

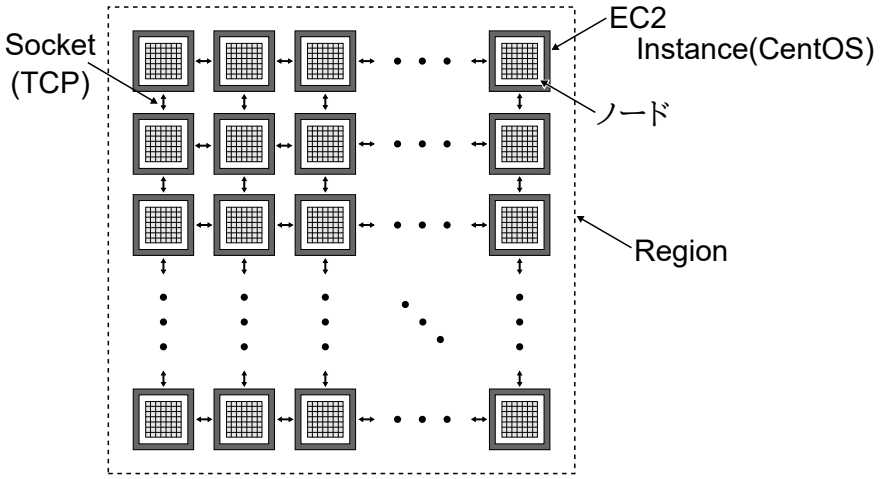


図 3.27: 性能計測対象とした基盤の構成

また，評価時には次の遷移則による単純な CA モデルを動作させた.

- 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）

- 近傍セルの状態が全て同一であれば，更新対象セルの状態もその状態に変化させる．
- 近傍セルの状態が全て同一でなければ，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，複数のノードを跨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上で都合が良いために今回の評価に用いた．図 3.28 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．

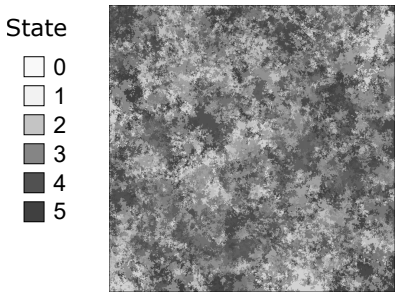


図 3.28: PCG モデルによって駆動される空間を可視化した例

3.6.2 評価 1 ：ノード数増加に伴う負荷変動

理論検証

ノード数の増加に伴い，各ノードに拡散されてくるサブワールドデータの量は増加するが，その増分は次第に漸減し，ある数を超えた時点でゼロになることが予想される．説明のために 1 次元で結合したノードでこの状況を示す．この例では，サブワールド（0）～サブワールド（9）の拡散半径を 16, 8, 4, 2, 1, 1, 1, 1, 1, 1 とする．まず，ノードが 1 台のみ存在している場合，当然ノードに拡散されてくるデータ量はゼロである．ここに 2 台のノードを追加し，図 3.29 の 1 のように，はじめのノードの左右に接続すると，中央のノードは左右のノードから全 LOD 分のサブワールドデータを受け取ることになり，このセル数の量は合計で $2 \times (1 + 1 + 1 + 1) \sum_{j=0}^{9} 2^j$ となる．さらに，図 3.29 の 2 のように左右にノードを追加した場合，これらのノードと中央のノードの距離は 2 となり，拡散半径が 2 よりも多い LOD 分のサブワールドデータが追加で拡散されてくることになる．この増分は $2 \sum_{j=0}^{3} 2^j$ となる．同様にノードを追加していくと，中央のノードとの距離が 4 となる時点の増分は

$2\sum_{j=0}^{2} 2^j$（図 3.29 の 3），また，距離が 8 では $2\sum_{j=0}^{j=1} 2^j$，距離が 16 の時点の増分は 2^0 となり，それ以上の距離での増分は，図 3.29 の 4 で示すように，0 になる．

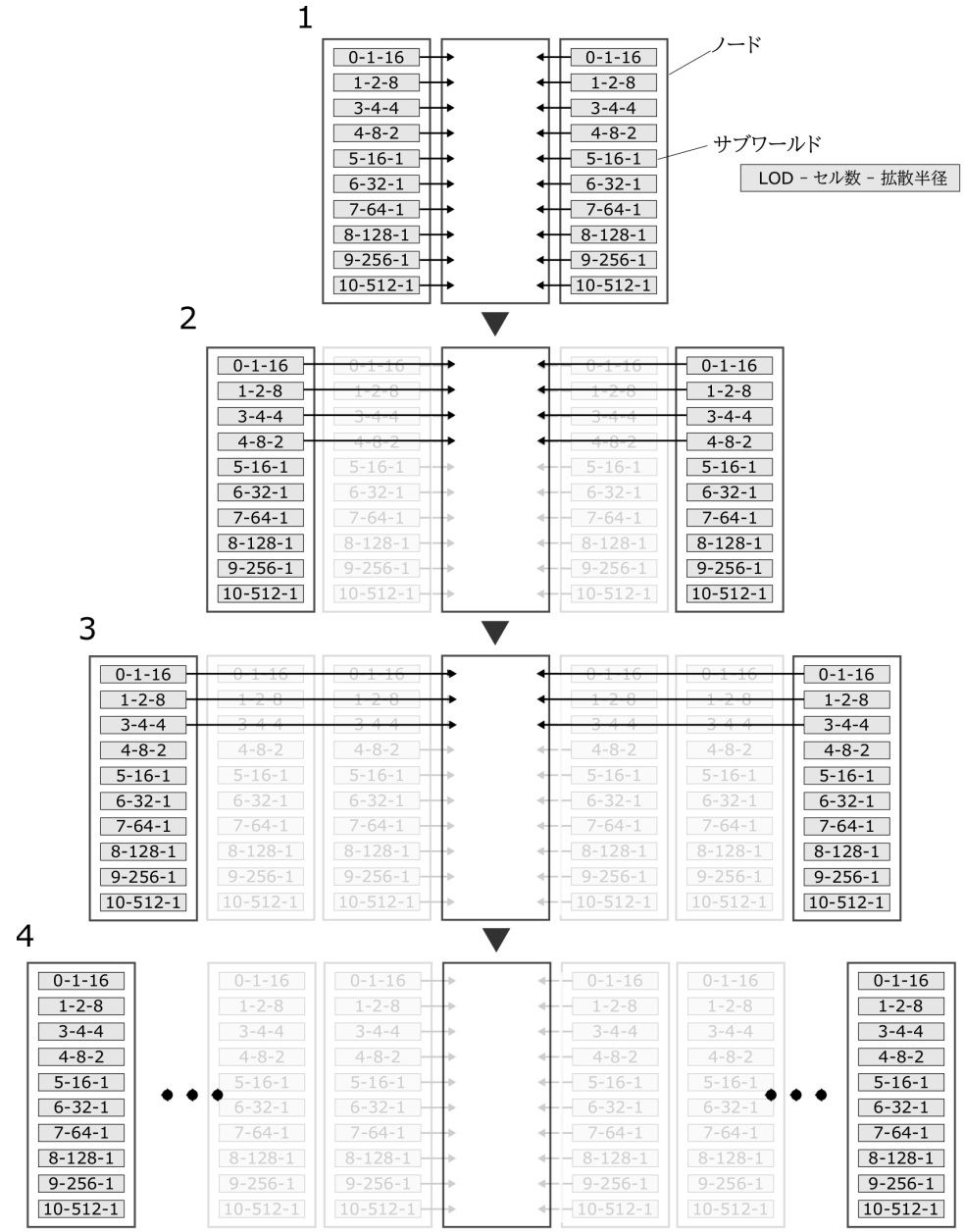


図 3.29: ノードに拡散されてくるデータ量の説明（簡単のため 1 次元で説明）

計測による検証

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リソース消費量を検証，計測する．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．ノードが保持するサブワールドのサイズの 1 辺の長さは 2^9 とした．したがってデータピラミッドには 10 個の異なる LOD を持つデータが含まれることになる．LOD 切替セル数（a_{swt}）は 2^{10}，サブワールドを構成するセ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，セルの更新，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 t3.micro（vCPU:2，メモリ:4GiB）を用いた．

評価 1 の計測環境を図 3.30 に示す．なお，CPU 消費率，ネットワーク帯域消費量は，CPU 消費率やディスク I/O，ネットワーク帯域消費などのリソース使用状況を 1 秒毎にモニタできるツールである Dstat[4]，メモリ使用量は Java 仮想マシンの動作状況を確認できるツールである jstat[5]を各ノード上で実行することによって計測した．計測間隔はいずれも 1 秒である．

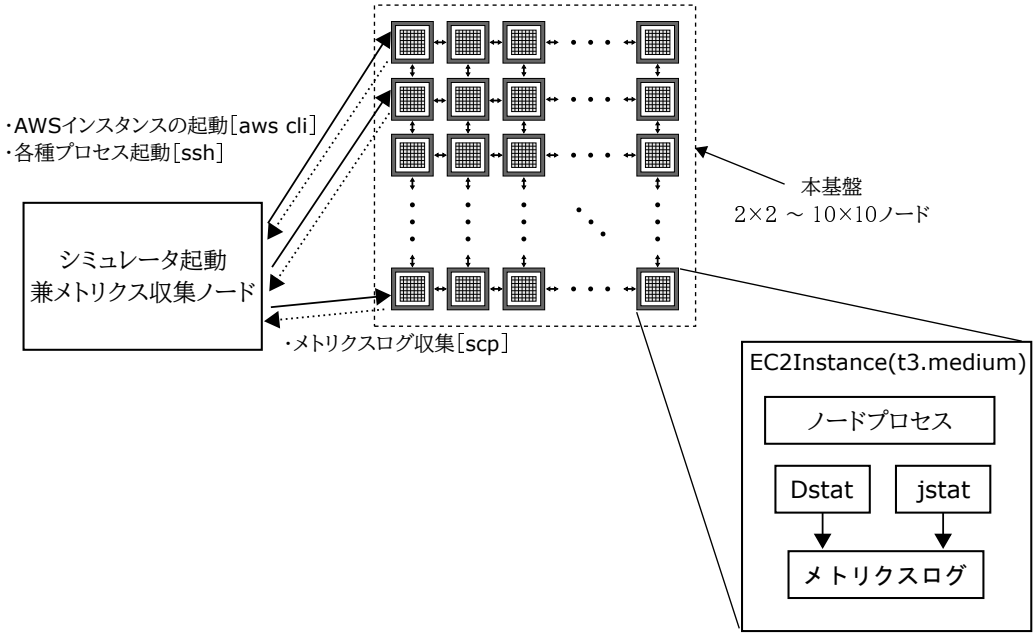


図 3.30: 評価 1 の計測環境

結果と考察

評価 1 の計測結果を図 3.31 に示す.（詳細な結果データは付録 A.1 に示す.）

[4]http://dag.wieers.com/home-made/dstat/

[5]https://docs.oracle.com/javase/jp/6/technotes/tools/share/jstat.html

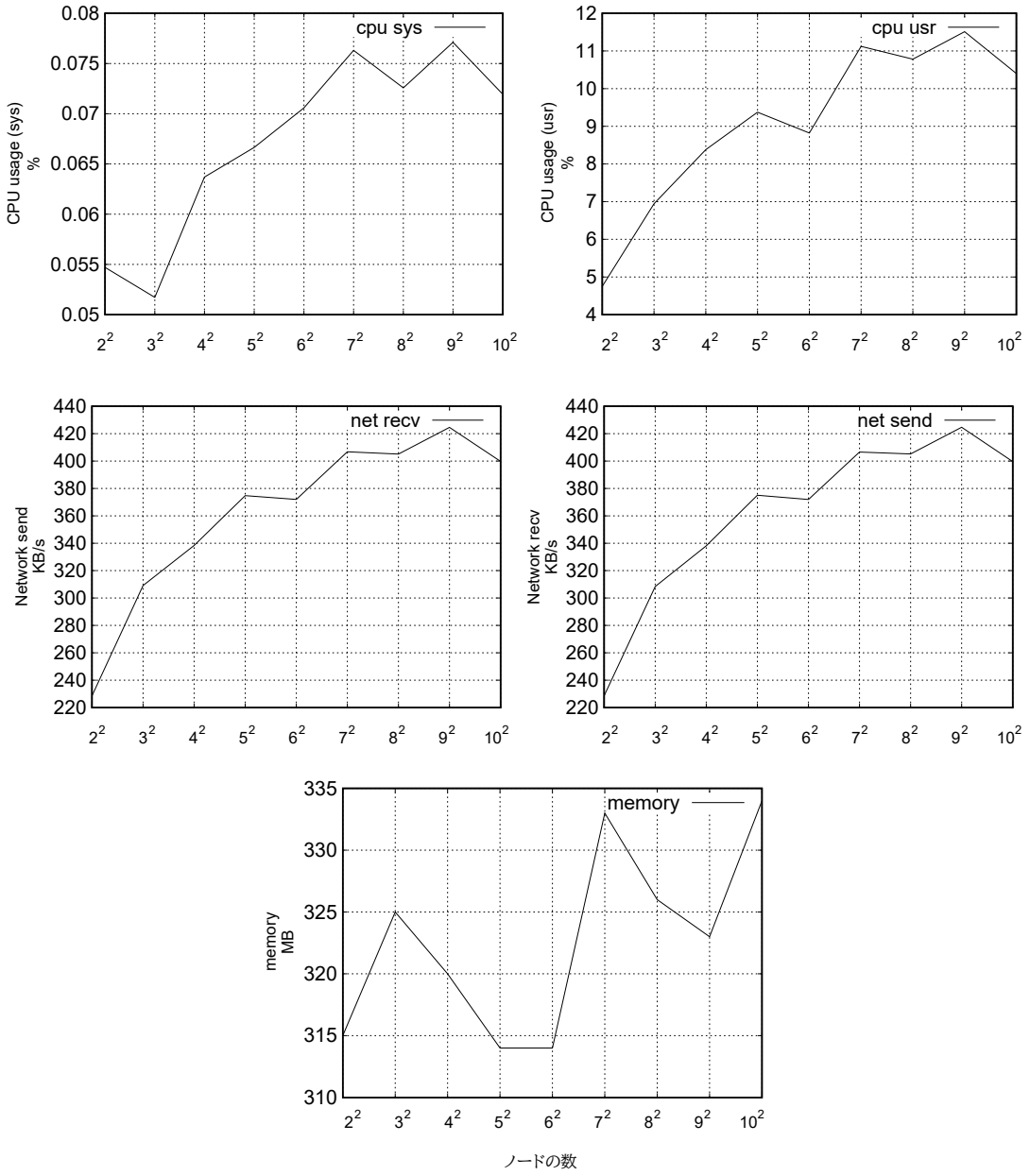


図 3.31: 評価 1 の計測結果：ノード数増加に伴うノードの負荷変動

図 3.31 はワールド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．メモリ消費量は Java のヒープメモリ全体サイズを 950MB に設定した上で消費量を計測している．また，今回の実験で用いた CPU 利用率の内訳の中で最も大きなものは，各ノードが保持するサブワールドデータを送受信するための Zip 圧縮／解凍処理であり，これが全体の 8 割程度を占める．次に大きなものはサブワールドデータをラスタ的に近接相互作用計算を行って更新する処理と，そこから LOD の低いデータを生成する処理であ

り，これが全体の 2 割程度を占める．グラフを見ると，ノード数が 7^2 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増加傾向にある．これはワールドの広さの増大に伴って，各ノードに対して，そのノードを中心にしたより広い範囲のノードのサブワールドデータが拡散されてくることが原因である．一方でノード数がそれより増えても全てのリソース消費量に大きな増加が無くなる．これはワールド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．メモリ利用量については図内のグラフでは変動が大きく，一見してこのような傾向は確認できないが，これは Java の GC の影響で頻繁にメモリ消費量が変動する状況を平均した値であることに起因する．実際には付録 A.1 で示すグラフから，ノードの全体数にかかわらずヒープメモリの消費量の上限は 650MB 前後に抑えられていることが確認できる．これらの計測結果によって，理論検証で予測した，ノード数が一定数を超えると，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挙動を確認できた．

なお，ワールドを広くするために，単一ノードの性能を高めて，保持するサブワールドデータのサイズを増やすことも考えられる．例えばエリアあたり AWS のインスタンス m5.large（vCPU:2，メモリ:8GiB）を 16 台で担当している構成を，m5.2xlarge（vCPU:8，メモリ:32GiB）を 4 台，あるいは m5.8xlarge（vCPU:32，メモリ:128GiB）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 128GiB となり，時間当たりの稼働費用も変わらない[6]．しかし，空間の広さを運用開始後も柔軟に少しずつ拡げていくためには，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．しかしながら，空間面積当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サブワールド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 LOD 切替セル数 a_{swt} 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．したがって実運用ではサービスの要件に応じてその都度適切なノード性能を選定することになる．なお，ノード性能のボトルネックはサブワールド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良い．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．

[6] 2022 年 11 月現在

3.6.3 評価２：クライアントからのリクエスト数増加に伴う負荷変動（従来方式との比較）

理論検証

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 3.32 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セル数に比例すると仮定してその比較を行った．

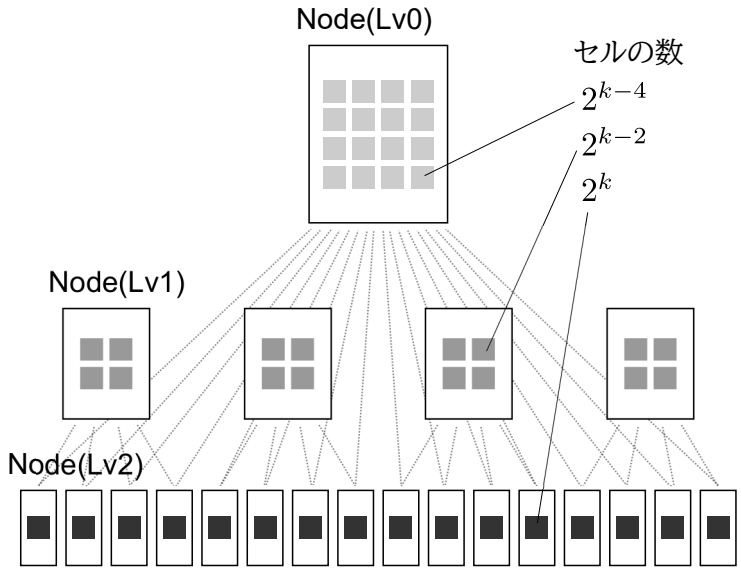


図 3.32: 代表サーバ方式の簡易モデル

図 3.32 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 のデータを更新し，Lv1 ノードに対して LOD を 1 つ下げた 2^{k-2} 個のサブワールドデータを，また，Lv0 ノードに対しては LOD を 2 つ下げた 2^{k-4} 個のデータを送信する．同レベルのノードに対してデータ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上で，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時に，各レベルのノードが

送受信するセル数はリクエスト数に確率的な係数を掛けて概ね式（3.13）の通りとなる．

$$
\begin{aligned}
\text{Lv0}: &\quad \underbrace{16\times 2^{k-4}}_{recv} + \underbrace{\frac{1}{3}\times 16\times 2^{k-4}\times x}_{send} \\
\text{Lv1}: &\quad \underbrace{4\times 2^{k-2}}_{recv} + \underbrace{\frac{1}{3}\times\frac{1}{4}\times 4\times 2^{k-2}\times x}_{send} \\
\text{Lv2}: &\quad \underbrace{2^{k-2}+2^{k-4}+\frac{1}{3}\times\frac{1}{16}\times 2^{k}\times x}_{send}
\end{aligned}
\tag{3.13}
$$

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セル送受信数は，

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

とおいて，

$$\underbrace{3\times\alpha}_{recv} + \underbrace{3\times\alpha + \frac{1}{3}\times\frac{1}{16}\times\alpha\times x}_{send} \tag{3.14}$$

となる．式（3.14）は，比較対象と条件を揃えるため，ノードが保持するセルの LOD は 3 段階とし，それぞれのサブワールドのセル数を $2^k, 2^{k-2}, 2^{k-4}$，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セ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，各ノードの送受信セル数変化を予測したグラフを図 3.33 に示す．

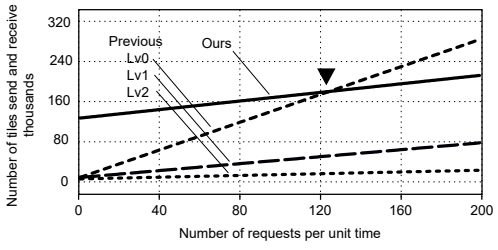


図 3.33: リクエスト数増加に対するセル送受信数変化傾向の予測

図 3.33 は，リクエスト数が少ない場合は提案方式のノードのセ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 ノードのセ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．

計測による検証

図 3.32 で示した代表ノード方式による従来方式を $k = 12$ とした上で，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．ノードに対するリクエスト負荷はクライアントサーバシステムの負荷テストを実行するツールである JMeter[7]により生成した．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，ノードには t2.large（vCPU:2，メモリ:8GiB）インスタンスを用い，負荷生成サーバには m5a.24xlarge（vCPU:96，メモリ:384GiB）を用いた．図 3.34 に評価 2 の計測環境を示す．また，図 3.35 に JMeter による負荷生成時の構成や設定を示す．

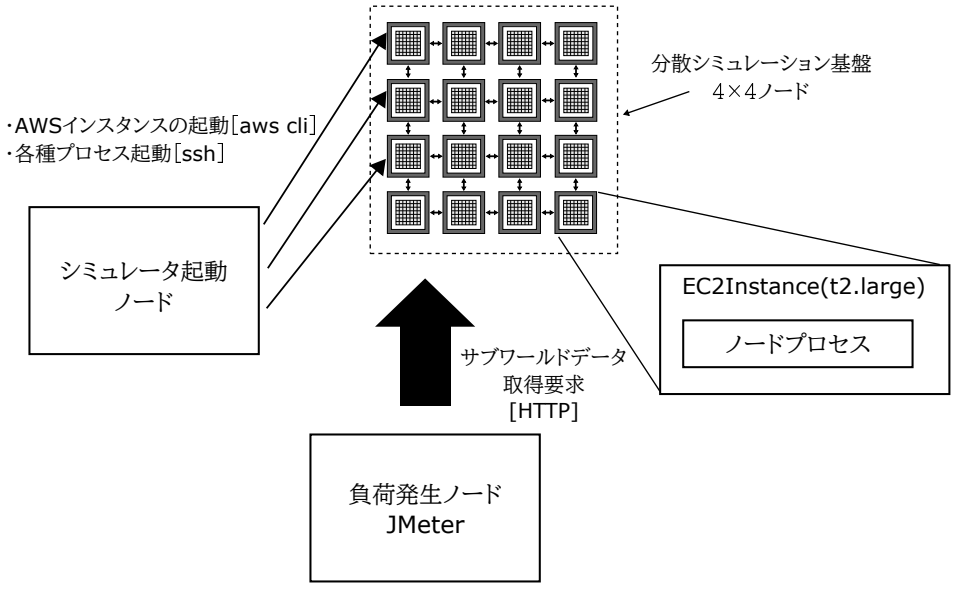


図 3.34: 評価 2 の計測環境

[7]https://jmeter.apache.org/

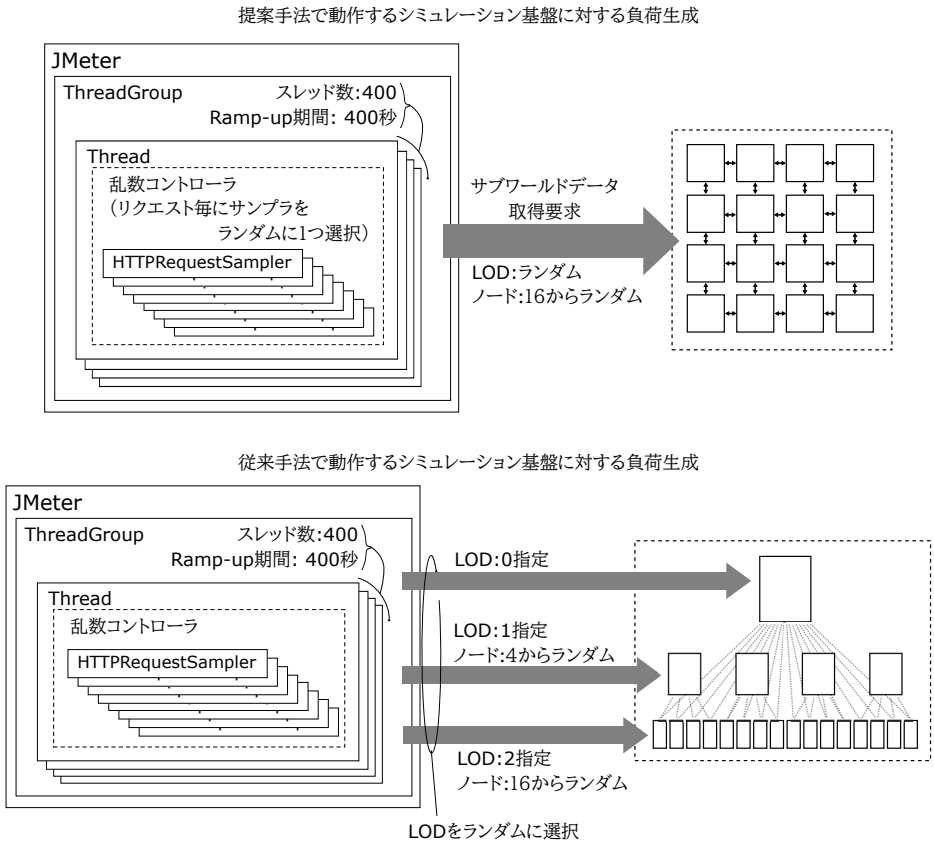


図 3.35: JMeter による負荷生成

結果と考察

図 3.36 に，シミュレーション結果を示す．図 3.33 では縦軸がノードあたりのセル送受信数，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．

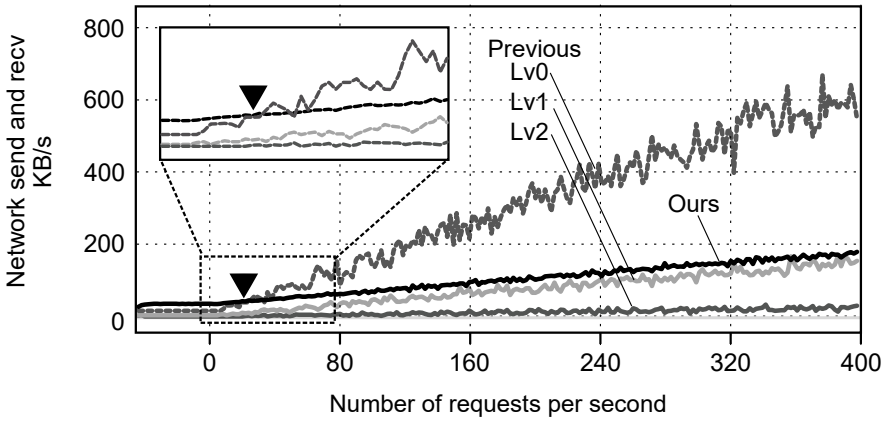


図 3.36: リクエスト数増加に対するネットワーク送受信量変化

この図は，各ノードの負荷増加傾向が概ね前節で述べた予測通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，予想通りリクエスト数の増加に伴って，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 3.36 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時の秒間リクエスト量は，図 3.33 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．

3.6.4 評価３：クライアントからのリクエスト数増加に伴うレスポンス時間の変動

リクエスト数の増加に対するレスポンスの遅延を評価するため，提案方式で動作するシステムに対して，テストクライアントからアクセスを行い，データを取得するまでの時間の計測を行った．全体ノード数は 3^2 で固定し，前節で示した方法と同様に，JMeter にてクライアント数を 10 台から 100 台まで変化させて並列に動作させて計測した．各クライアントは 9 ノードの中からランダムで選んだノードに対して保持する空間データを全て取得するリクエストを送信し，レスポンスを得られた後 1 秒待機し，再度別のノードをランダムに選択しリクエストを送信することを 100 回繰り返し（毎秒 1 リクエスト，合計 100 秒），全テストクライアント分の試行結果からパーセンタイル値を算出した．

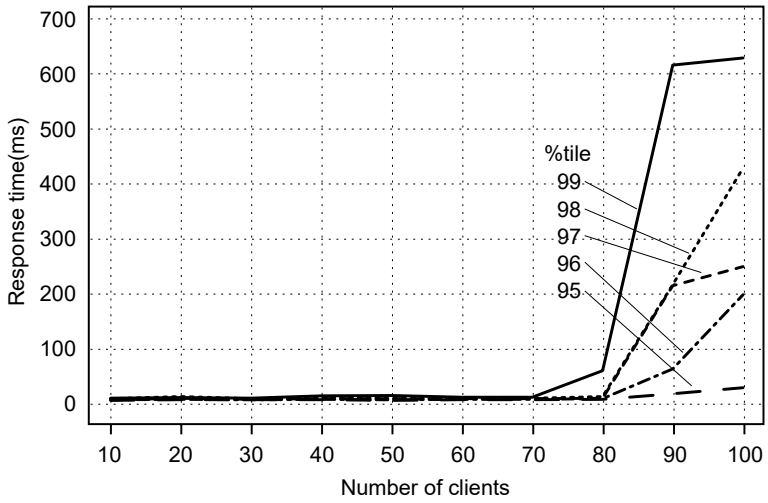


図 3.37: リクエスト数増加に対するレスポンスの遅延

結果と考察

図 3.37 は，ノードに対してデータのリクエストを依頼し，そのレスポンスを得るまでの時間のパーセンタイル値（計測結果を値の低い順に並べ，低いから側から指定割合分に位置する値）を示したものである．横軸はクライアントの数を示しており，今回はクライアントが毎秒リクエストを送信しているので秒間のリクエスト数と見ることできる．秒間リクエスト数が 70 ／秒まではレスポンス時間は 20ms 内に抑えられている．80 ／秒の時点で少しレスポンス時間が増加しているが 99 パーセンタイルで 100ms 以内に収まっている．これ以降は次第に遅延が増加していく．

この結果から，本検証時のノード 9 台という構成のシステムについては，80 クライアントが毎秒データをリクエストするような状況下では，100ms 以内にレスポンスを得ることができ，これはユーザが空間をインタラクティブに可視化するにあたって支障無い範囲のレイテンシであると言える．なお，今回の評価ではノードの性能が低い（仮想 CPU 数が 1）のインスタンスを用いていることに加え，クライアントからのリクエストを処理するスレッド数を 1 として実装している．したがって，仮想 CPU の数を増やした上でスレッド数をチューニングすることで，このレイテンシはさらに改善することが可能と考えている．

3.6.5 評価４：データのレプリケーション遅延

データのレプリケーション遅延を評価するため，ノードが生成したデータが他のノードに到着するまでに要する時間を計測した．ノードの台数は 10^2 台で固定した上で，評価 1 における実験環境と同じ条件で動作させたシステムを対象として，ノード間のデータの流通に要する時間をノード間のホップ数別に集計した．なお本評価では，実運用ではノードが異なるデータセンタに分散して配置される場合を考慮し，実験用のノードの AZ 配置を図 3.38 に示すように 3 パターンに分け，それぞれの配置パターンで計測を実施した．図 3.38 において，網掛けで示している領域に属するノードは，網掛けではない領域に属するノードとは異なる AZ に配置している．パターン 1 は評価 1 と同様に全てのノードを同一の AZ に配置する構成，パターン 2 はノードを 5^2 台のグループに分けて異なる AZ に配置する構成，パターン 3 はノードを 2^2 台のグループに分けて異なる AZ に配置する構成である．実験環境において Ping コマンドによって事前計測した AZ 内の通信と，AZ を跨いだ通信の差異を表 3.3 に示す．

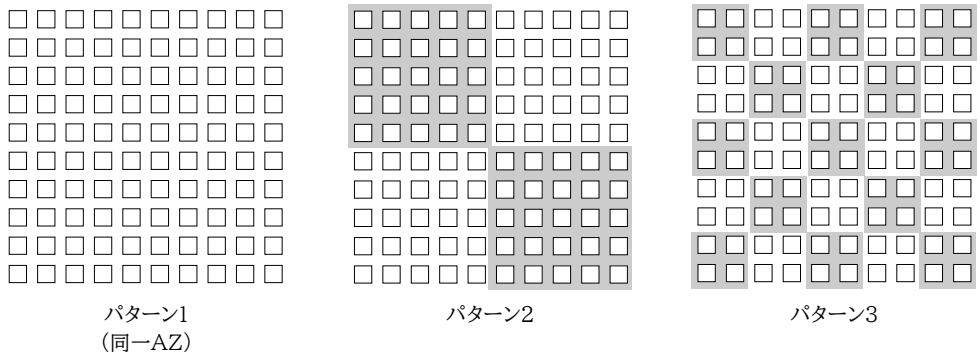


図 3.38: ノードの配置パターン（網掛け部分が異なる AZ）

	AZ 内 RTT（ms）	AZ 間 RTT（ms）
min	0.181	0.972
avg	0.227	1.023
max	5.617	7.548

表 3.3: AZ 内通信と AZ 間の通信時間

結果と考察

図 3.39〜3.41 に，それぞれの配置パターンにおいて，ノードが毎秒生成するデータが他のノードに到着するまでの平均時間とその標準偏差をデータのホップ数ごとに示した．いずれの配置パターンにおいても，データのホップ数に比例して到着に要する時間が増加する傾向が確認できた．これはデータが他のノードを介してマルチホップで届けられることを考えると当然の結果である．また，今回の実験条件では，全てのノードは毎秒非同期に隣接ノードに対してデータを送信しており，結果として，ノードが生成したデータが他のノードに到着するまでにかかる時間は概ねホップ数の半分程度の秒数となることがわかった．

なお，今回の実験では，ノードの配置パターンによるデータ到着時間の有意な違いは確認できなかった．これは，今回の実験でノードが送受信するデータの量が高々数百 KB/s 程度であることからネットワーク帯域に十分な余裕があり，AZ を跨ぐことによる通信遅延の影響を受けなかったことが理由と推測される．

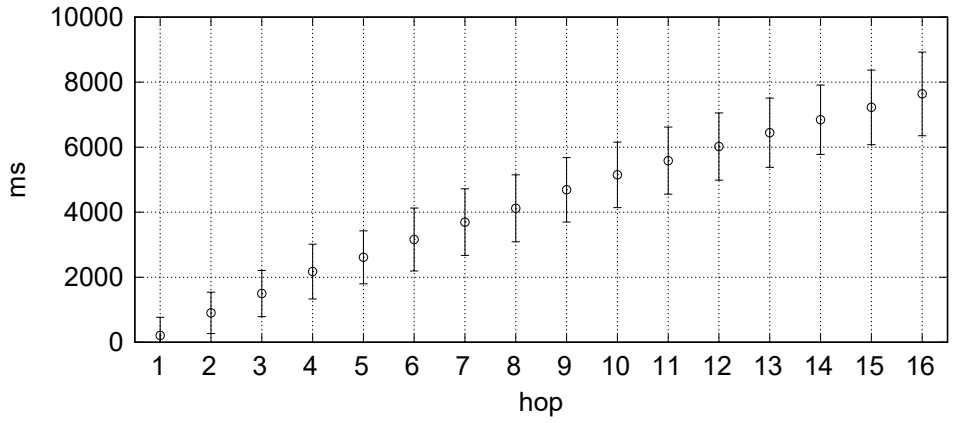


図 3.39: レプリケーションに要する時間（配置パターン 1 ）

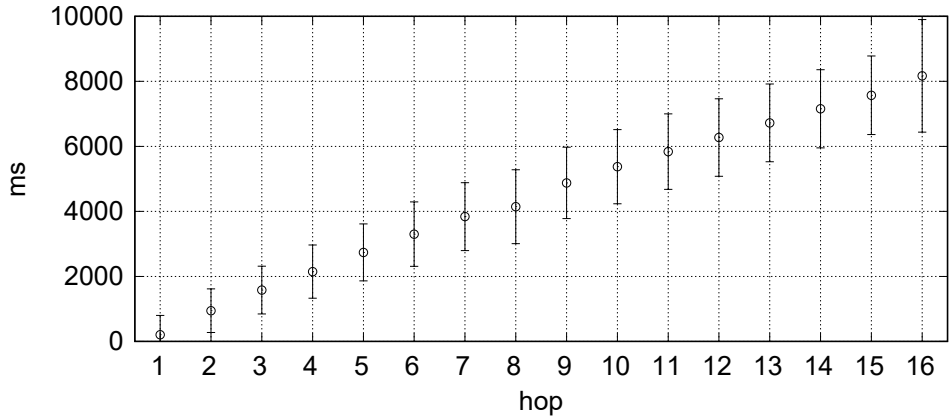


図 3.40: レプリケーションに要する時間（配置パターン２）

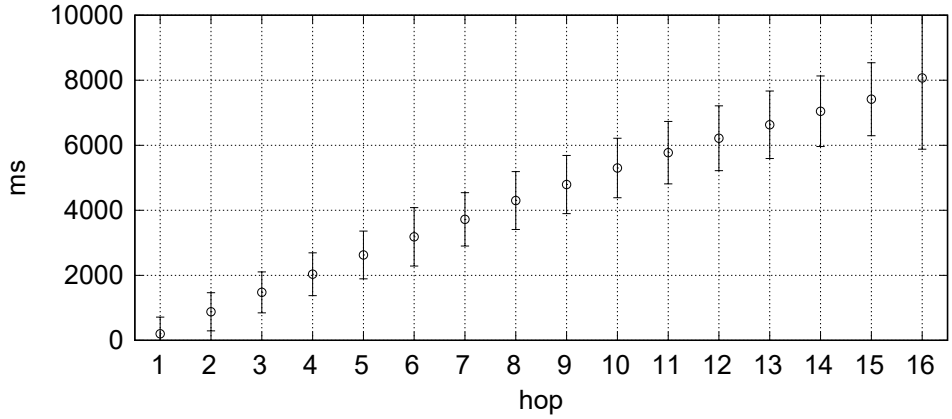


図 3.41: レプリケーションに要する時間（配置パターン３）

3.7 まとめと今後の課題，制約

変化する空間の広域な可視化を目的として，保持するデータを拡散的に周囲のノードへ複製することで，可視化を効率良くおこなうための理想的なデータ分布を生成する自律分散システムの提案と実装，評価を行った．評価の結果，ノード数の増加に応じたリソース消費負荷が一定に抑えられること，リクエスト数に応じたネットワーク送受信量が，秒間リクエスト数が多い時に従来方式よりも低く抑えられること，また，クライアント数に応じたレスポンス時間の増加傾向と，データのレプリケーションに要する時間を確認した．

3.7.1 提案方式における制約

本基盤は空間データのサイズを大規模にすること，および，その空間を多様な縮尺で可視化することを重視する一方で，データの一貫性や冗長性，耐障害性は妥協している．本節では，提案方式を用いることで，アプリケーションに生じ得る制約や不整合について述べる．

空間データの一貫性や整合性

提案手法はノード間でデータを非同期的，かつマルチホップで段階的に複製するため，ノード毎にデータ複製のタイミングに差が生じ，保持するデータの内容が完全には一致しない．このため，状況に依ってはクライアントがデータを取得するノードによって，異なる内容が返される可能性があり，また，同じノードからデータを取得していても，アクセスする度に異なる内容が返される可能性がある．具体的には，空間を可視化した際に，特定の領域の内容が，その周辺領域と比べて一時的に数世代（シミュレーションの時間発展計算ステップ）分過去の内容が表示される可能性がある．なお，このような一時的な不整合が生じて，見た目に違和感が生じたとしても，しばらく（毎秒データ複製のためのデータ送信が行われるとしたら数秒）待てばデータの複製は完了し，その状況は解消される．3.1 で述べたゲームを想定したアプリケーションでは，これらの不整合によってゲームの魅力が損なわれることはないと考えている．しかし，コネクテッドカーから取得したアップロードされたデータにもとづき交通状況を把握するようなアプリケーションにおいて，例えば特定のエリアの混雑度が一定以上であればドライバのカーナビやスマートホンなどの端末に対して自動的に注意喚起通知を行うようなサービスを行う場合は，空間データの内容に基づき即座に混雑度を閾値判定して通知を発呼してしまうと，過去の情報にもとづいた誤通知が発生する可能性がある．そのため，このような用途では別途通知の発呼を行う際に，一定時間待ってデータの内容が変化しないことを確認してから通知を送るといった対策が必要になる．

また，提案手法はアクセスしているノードから，ノードネットワーク上の距離が遠いノードから送信されてくるデータほど到着に時間がかかる（3.38 節で述べた評価実験の結果より，ノードネットワーク上のノード間のホップ数の半分程度の秒数を要することがわかっている）．また，広域表示用の LOD が低いデータほど大きな拡散半径が設定され，より遠くのノードからデータが届けられる．これらのことから，システムの利用者が，仮想空間を広域に表示しようとするほど過去の内容が表示される制約がある．ただし 3.1.2 節で述べ

たように，特定の狭い範囲を詳細に可視化している時と比べて，広い範囲を俯瞰的に可視化している状況では，遅延に対する許容幅が大きいと考えているため，この制約は想定する利用用途では大きな問題にならない場合が多いと考えている．

ただし一方で，2.1.2 節で述べたように格子状に接続されているノードのネットワークに対してランダムにショートカットを加えることで，ノード間の ASP を抑え，ノード間のデータ流通に要する時間を低減することも可能である．これには，格子状に接続されている各ノードに対して，前節で述べた拡散半径のうち最大のもの（つまり r_0）の距離範囲内に存在するノードを対象にランダムにリンクを追加し，データを届ける必要のない拡散半径を超えた範囲に存在するノードと接続がなされないようにした上で ASP を短縮すればよい．

シミュレーションの精度

本基盤上で空間の時間発展シミュレーションを実行する場合は，動作モードによってはノード同士が同期をとらずに個々のクロックに従って独立してシミュレーションを実行する．このため，空間上の場所に依ってシミュレーションの進行状況に差が生じ，シミュレーションに誤差が生じる可能性がある．これが許容できるかどうかはアプリケーションによって異なる．例えば物理現象を精緻に再現しようとするアプリケーションでは，このような誤差は問題となる．一方で，高い精度が求められない用途では，このような誤差は問題にならないと考えている．また，ノードが保持する空間領域の内部については，全てのセルの更新は同期的に行われるため，シミュレーションの進行は揃う．さらに，ノード間の境界に位置する領域では進行に差が生じる可能性があるが，ノード境界に位置するデータの更新は，ノード間でデータを交換した上で行われるため，それぞれの領域が互いに関係なく独立して進行するわけではない．3.1 章で述べたゲームアプリケーションを想定すると，ノードの境界に位置する領域以外の場所ではシミュレーションの進行が揃うため特にこの制約の影響を受けず，また，ノードの境界を跨る領域であっても，仮想世界で生じる例えば火災などの事象の空間的な連続性が断絶されるわけでないため，ゲームのユーザが見た目に違和感を感じるようなことは少ないと考えている．

ノード障害時のデータ読み書きと計算進行

ノードに何かしらの障害が発生してプロセスが停止した場合，そのノードが保持するデータは参照できるが，書き込みができなくなる．これは読み取り負荷を分散するためのデータレプリケーションはノード間で随時行われるため，当該ノードが仮に停止したとしても

そのデータは近隣ノードが保有している一方で，データの内容を更新するための処理は当該ノードのみしか行わないためである．同じ理由で，本基盤上で時間発展シミュレーションを実行する場合，ノードが停止すると，その領域のシミュレーションの進行は停止する．また，2.1.1 節の図 2.3 に示したようにクライアント側で時間発展シミュレーションを行う場合は，空間上の領域に依って時間の進行が大きく異なる可能性が高くなる．したがって，例えばゲームへの応用を想定した場合には，これらの不整合を許容するか，あるいはゲームのルールとして積極的に取り込んだ企画デザインを検討する必要がある．

3.7.2 今後の課題

今後の課題として最も大きなものは，運用にかかる作業を減らすため，障害復旧や負荷の偏り等に対する対応を自動化することである．これについては 6.2.4 節で述べる．ここではその他の課題について述べる．

可視化できる仮想空間の範囲制限の解消

提案方式は，仮想空間の広さの拡大に対するノードの負荷を一定に抑える代償として，空間を俯瞰できる範囲が限定されることになる．この制約を解消し，空間の全域俯瞰を可能とする方式が考えられる．この方式については A.4 節にその考えを述べる．

最適配備構成の見積方法の検討

3.6.2 節でも述べたように，提案方式はアプリケーションの要件に応じて，ノードのリソース性能と，ノードに割り当てる空間の広さを決定する必要がある．したがって，アプリケーションの利用する通信帯域や，空間データ更新にかかる処理の負荷量などに応じた，最適なノードの配備構成の事前見積もり方式を確立する必要がある．

正方格子以外による空間データ表現

本稿では空間データ正方格子に区切ることを前提としていた．しかしながら，格子間の相互作用計算を単純化できる等の理由で，例えば六角格子などの正方格子以外の表現によって仮想空間を構成したほうが望ましい場合がある．また，惑星（球）の上で仮想空間を表現する場合も，球の極付近に発生する格子の歪みを抑えるためには正方格子ではない方法で空間を表現する必要がある．したがって，正方格子ではない方法で空間を構成した場合

の，空間の粗視化方法や，そのデータの流通方法，および，その流通方法に合わせたノード同士の理想的な接続形態を検討する必要がある．

第4章　近傍領域のモデル化と高速計算手法

4.1　背景と目的

2.2 節で述べたように，仮想空間上でより臨場感のあるインタラクションを可能とするため，仮想空間上における特定地点の近傍領域の概念が提唱されている．しかし，例えば特定地点から一定の距離を半径とした円領域などの単純な形状による近傍定義は，その近傍領域が空間上の構造物を突き抜けてしまい，この近傍を介してインタラクションを行うと，違和感の原因となる．そのため，直感的なインタラクションを実現するためには，空間上の構造物を考慮して近傍の形状が変化する必要がある．

本章で述べる研究の目的は，仮想空間上の構造物に応じて柔軟に形状を変える近傍のモデル化と，その高速計算手法を確立することである．近傍は，多様なアプリケーションから汎用的に利用することを想定しており，構造物や近傍の具体的な意味付けはアプリケーションに依存して決まる．代表的な利用例としては，MMORPG において，図 4.1 に示すように，ユーザが操作するシミュレーション空間上のキャラクタの近傍領域を，その周辺の石や岩，ブロック塀などの構造物として指定したセルの位置に応じて生成し，その領域内に存在するセルの属性（炎，氷，水など）の影響によりキャラクタの状況を変化させたり，あるいは自動的に動作する敵キャラクタの近傍領域を定め，その領域内に他のキャラクターが侵入した時点で攻撃を開始するといった判定領域として利用するといった使い方が考えられる．また，例えば洞窟内などの暗い場所で，キャラクターの周辺のみ洞窟の構造にそって段階的に照明の強度を明るくするような視覚効果のために利用することもできる．

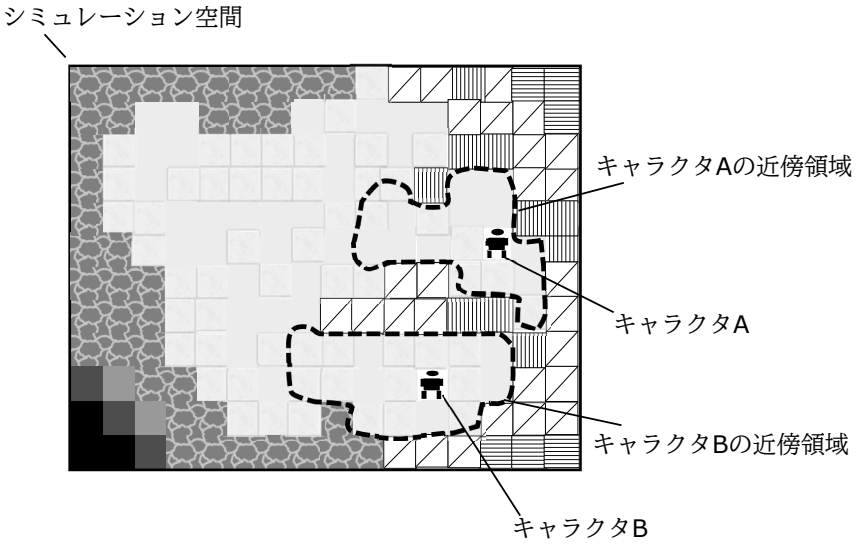


図 4.1: MMORPG における近傍領域の応用イメージ

4.2 先行研究

仮想空間において，注目地点からの近傍を空間内の構造物を考慮して決定する方法として，注目地点と別のある地点の 2 点間の最短経路パスを生成し，その長さが閾値以下であることをもって，その点が注目地点の近傍範囲内に位置すると判定する方法が考えられる．Hershbaerger [62] は，障害物を含む平面上の 2 点の最短経路を $O(n \log n)$ で計算する方式を示している．Nilsson [87] は，障害物の頂点間を結ぶ直線によって構成される可視グラフから最短経路を求める手法を提案している．他にも，平面上での最短経路を求める計算幾何学的なアプローチは多数存在している．しかし，例えば仮想空間内で，注目地点に近い領域ほど照明を明るく照らし，遠ざかるほどそれを暗くしていくような視覚効果を実現する場合，計算幾何学的な方式を用いると仮想空間の多数の格子と注目地点が属する格子の間で最短経路パスを生成する必要があり，計算の回数が大量になり現実的ではない．

Zelinsky [113] は，ロボットの経路探索を行うための方式として，ロボットが行動する空間を格子に分割して距離変換 [136] を行うことで距離場を定義し，それをもとに経路パスを求める方法を示した．また，ダイクストラ法 [47] や，A-star アルゴリズム [61] を格子に適用し，同様に距離場を生成した上で最短経路を求めることができる．これらの方式で生成される距離場は，計算幾何学的なアプローチと比較し，空間内の多くの点と注目地点の距離が得られるため，それを用いて注目地点を中心としてそこから段階的に照明を暗くするような視覚効果が自然に実装できる．しかし本節で述べる近傍は，平面上における障害

物を考慮した注目地点からの経路距離のみにより定義されるものではなく，注目地点の周辺の障害物の取り囲み方や空間的な広がりも加味されて決定される「気配」のようなものを想定している．例えば図 4.2 で示すような空間構造の中で注目地点が部屋の中に位置している状況において，点 A と点 B は注目地点から等しい経路距離に位置しているが，点 A は注目地点と同じ部屋の中に位置しているため，点 B よりも多くの情報を注目地点から感じとるのが自然と考える．このような空間構造を考慮した近傍を定義できる距離場は提案されていない．

「気配」とは，何かしらの物理的な事象が空間上で伝播することと考えると，音響や光の伝播を，2.2 節で述べたレイベース法や波動ベース法などの手法による物理シミュレーションを実行することによって，それを模擬できる可能性がある．しかし，これらのシミュレーションの計算負荷は高く，後述するようにクライアント側で計算することを考慮すると，物理シミュレーションによる方法はバッテリなどの消費が大きくなり望ましくない．

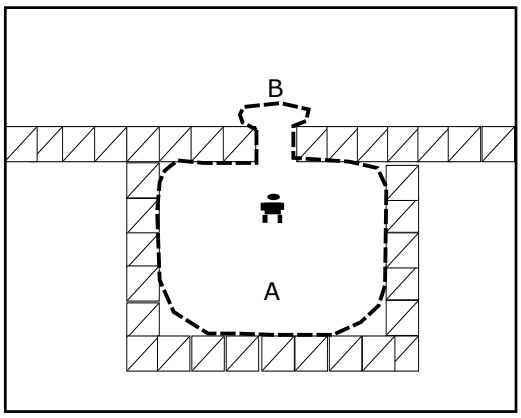


図 4.2: 「気配」としての空間構造を考慮した近傍のイメージ

4.3 近傍領域のモデル化

4.1，4.2 節で述べた議論を踏まえ，近傍領域の要件を次のように定める．

- 空間上の構造物に沿って柔軟に形状を変化させた領域を生成できる．このため，構造物が複雑に入り組んだような場所であっても，そこに適応して近傍を定義することができる．
- 位置の関数として，連続量として注目点からの近さが得られる．これにより，例えば洞窟の中のよう暗い状況下において，主人公の付近だけ照明を明るくし，主人公から離

れるにつれて徐々に照明を減衰させていくような視覚効果を施すことができる．なお，この近さは直線距離ではなく，空間上の構造物を考慮した衝突回避経路:Collision-free path 上の距離となるため，照明（光）の到達範囲は構造物に沿った形状となる．

- 図 4.2 で示したような，同じ場にいるほど近いとみなされるような自然な近傍を再現できる．

4.3.1 ポテンシャルによるモデル化

前節で述べた要件を満たすため，本提案では近傍領域をポテンシャルによってモデル化する．具体的にはラプラス方程式（式 4.1）の解となる調和関数場とその等高線によって近傍領域の形状を表現する．

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{4.1}$$

ポテンシャルは多く応用されている．ロボティクスの分野では，ロボットが自律走行する際の経路生成や電動車椅子の歩行者回避アシスト，ロボットアームの動作軌道生成のためにポテンシャルを用いることが多く提案されている [75,95,102,129]．また，ゲームやシミュレーションの領域では，乱戦シーンにおいて，複雑な条件分岐なしに複数のアバターを自律的に移動させたり，魚群行動を再現するためにポテンシャルが用いられている．[120,121,133]

図 4.3 にポテンシャルの例を示す．

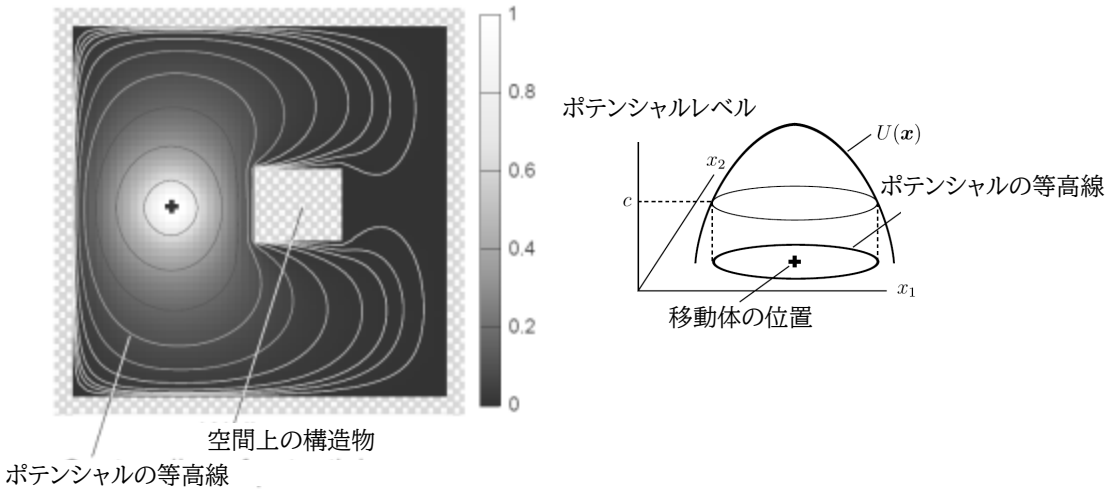


図 4.3: ポテンシャルの例

ポテンシャルは非圧縮性流体の円柱周りの速度の分布や，電荷点周りの電位の分布，熱媒質中の熱の分布などの定常状態を記述することでも知られており，空間上に何かしらの物や事象が存在するときに，それが周辺に及ぼす物理現象を自然に表現することが多い．このことから，シミュレーション空間上に存在する構造物の音や香り，光が物理的に届く範囲の近似的な表現（光や音や匂いが届く範囲はそれぞれ個別に物理モデルが存在しているが，物理的な精度よりも計算速度が速く，かつ見た目もそれらしい表現）としてポテンシャルを応用し，それを近傍領域として利用してインタラクションを行えば，多くの場合で見た目に違和感のない結果になると考えた．

4.4 近傍領域の高速計算手法

ポテンシャルを生成するためのラプラス方程式の定常解を数値的に求めるためには，ポテンシャル分布を生成する対象の空間を離散化してできる格子の数の元を持つ連立方程式を解く必要があり，これには時間がかかる．例えば 80×80 個に離散化した空間上でのポテンシャルを生成するために，6,400 元連立方程式を解く場合，ソルバとして Java Matrix Package（JAMA）[1] を CPU が 4 コアのマシン上で利用した場合の例で約 226 秒を要する．このように長く計算に時間がかかるとスムーズなインタラクションに支障が生じる．スムー

[1] https://math.nist.gov/javanumerics/jama/

ズなインタラクションのためには，刻々と変わる空間上の構造物の状況や，近傍を生成する対象物の位置にリアルタイムに追従してポテンシャルを生成する必要がある．見た目に遅延を感じない限界付近である 20FPS 前後のフレームレートで画面を更新しするためには，約 50 ミリ秒以内にポテンシャル生成を行う必要がある．大規模な連立方程式の求解の高速化手法は SOR 法（Successive Over Relaxation method）やクリロフ部分空間法などが存在しているが，これらは連立方程式の係数行列の特性（非零要素数の数等）に要件があったり，前処理が複雑であったりと，今回の問題には適さない．GPU（Graphic Processing Unit）を用いて計算を並列実行することでも，計算時間の短縮は可能である．しかし，ポテンシャルの生成はクライアント側で計算することを想定しているため，クライアントマシンの GPU の性能や搭載有無によって計算の時間が著しく変わることは避けたい．また，クライアントマシンはスマートフォンなどの携帯端末である可能性もあり，GPU の高頻度な利用による消費電力増大に伴うバッテリーの早期消費も避けなければならない．

上記の要件に対応して，本手法は，ラプラス方程式の境界値問題を近似的に解く手法である代用電荷法 [33, 70, 71] を応用してポテンシャルの計算を行う．代用電荷法は，ポテンシャル分布を計算する空間上に，幾つかの電荷点と，その電荷点から発生する電位の量を拘束する標本点を配置し，標本点で指定した値を満たすようなポテンシャル分布を近似的に計算する．代用電荷法は解くべき連立方程式の元数が標本点の数にまで減らすことができるため，計算時間を劇的に向上させることができる．代用電荷法によって得られるポテンシャル分布は近似値となるため，ラプラス方程式を解いた結果得られる厳密なポテンシャル分布とは一致しないが，ゲームなどへの応用を想定した見た目に違和感のない近傍領域を生成するという目的のためには精度は求められず，また，後述するように，代用電荷点では精度誤差が，近傍領域のために用いる箇所とは離れた標本点付近で大きくなる傾向があり，精度が問題となることは少ないと考えている．

4.4.1 代用電荷法を応用した近傍領域の近似計算

本節では代用電荷法を応用して注目地点の近傍領域を生成する方式について記す．代用電荷法は，標本点で定めた境界値を拘束条件としながら，電荷点を中心として分布する対数関数の重ね合わせによってラプラス方程式の境界値問題を近似的に解く方式である．すなわち，空間上で幾つかの点とその点における値を定めれば，その位置でその値を満たすようなポテンシャル分布を得ることができる．

本手法では，図 4.4 で示すように，注目地点周辺を取り囲むように仮想境界を設置し，その境界上での値を最大に，壁面境界での値を最小にすることによって，注目地点の位置を

ピークとして，そこから周辺の空間に沿って減衰するポテンシャルを生成する．

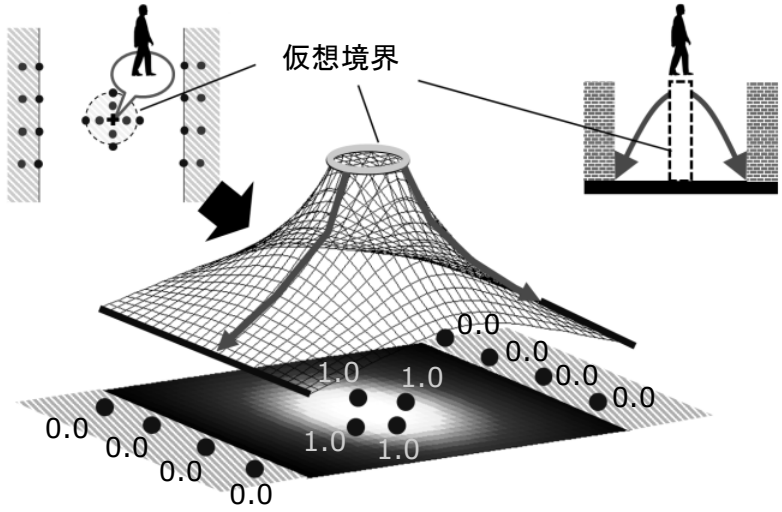


図 4.4: 代用電荷法を応用した近傍領域生成．空間構造物の壁面と，注目地点の周辺に仮想境界を定めて拘束値を設定する

具体的には図 4.5 に示すように，近傍を計算する注目地点の周辺と，空間構造物の壁面に沿って電荷点を，その外側に標本点を配置する．そして，それらの標本点の値について，注目地点の周辺に沿って配置した分については最大値，壁面に沿って配置した分については最小値を設定する．

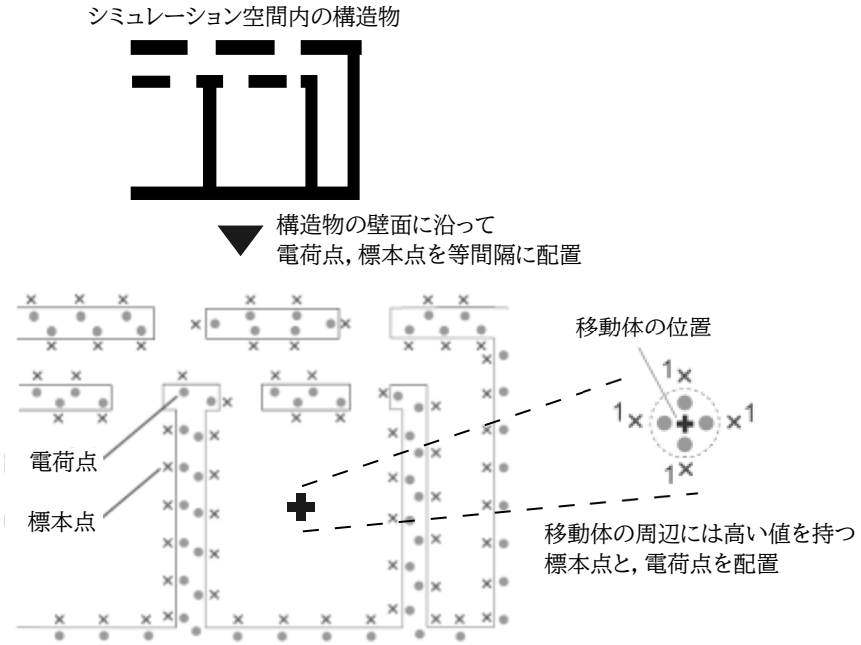


図 4.5: 電荷点と標本点の配置方法．空間構造物の壁面と，注目地点の周辺に沿って電荷点と標本点を配置する

電荷点と標本点の位置は，シミュレーション進行によって時間発展変化する空間構造物の配置と，その中をユーザの操作などに応じて動き回る注目地点の位置に応じて動的に変える必要がある．そのため，ノードがシミュレーションを進行させるのと並行して，空間構造物に沿った点の情報を生成してクライアントに提供し，クライアントではそれにユーザが操作する注目地点を囲む点の情報を併せてポテンシャルを計算する．

図 4.6 の右に，本手法によって生成したポテンシャルの例を示す．左には同じポテンシャルを従来の差分法によって生成した例を示す．また，図 4.7 に，より複雑な構造の空間内を移動する対象に対してポテンシャルを生成した例を示す．この図の右上で示した近傍形状を見ると，同じ部屋の中に広がって部屋の外へは僅かに滲み出ている程度であり，空間構造を考慮して同じ場所にいるほど近いとみなされるという，先述した要件を満たしていることがわかる．

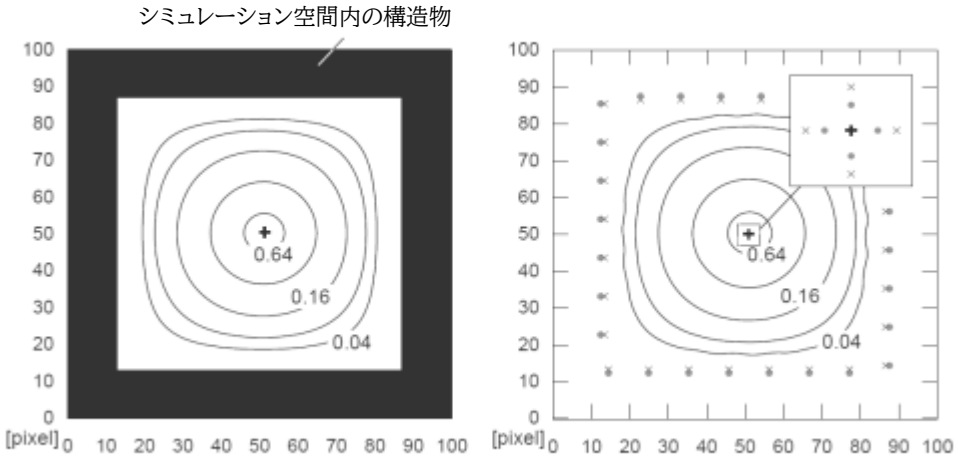


図 4.6: 生成したポテンシャルの例：左が差分法，右が本手法

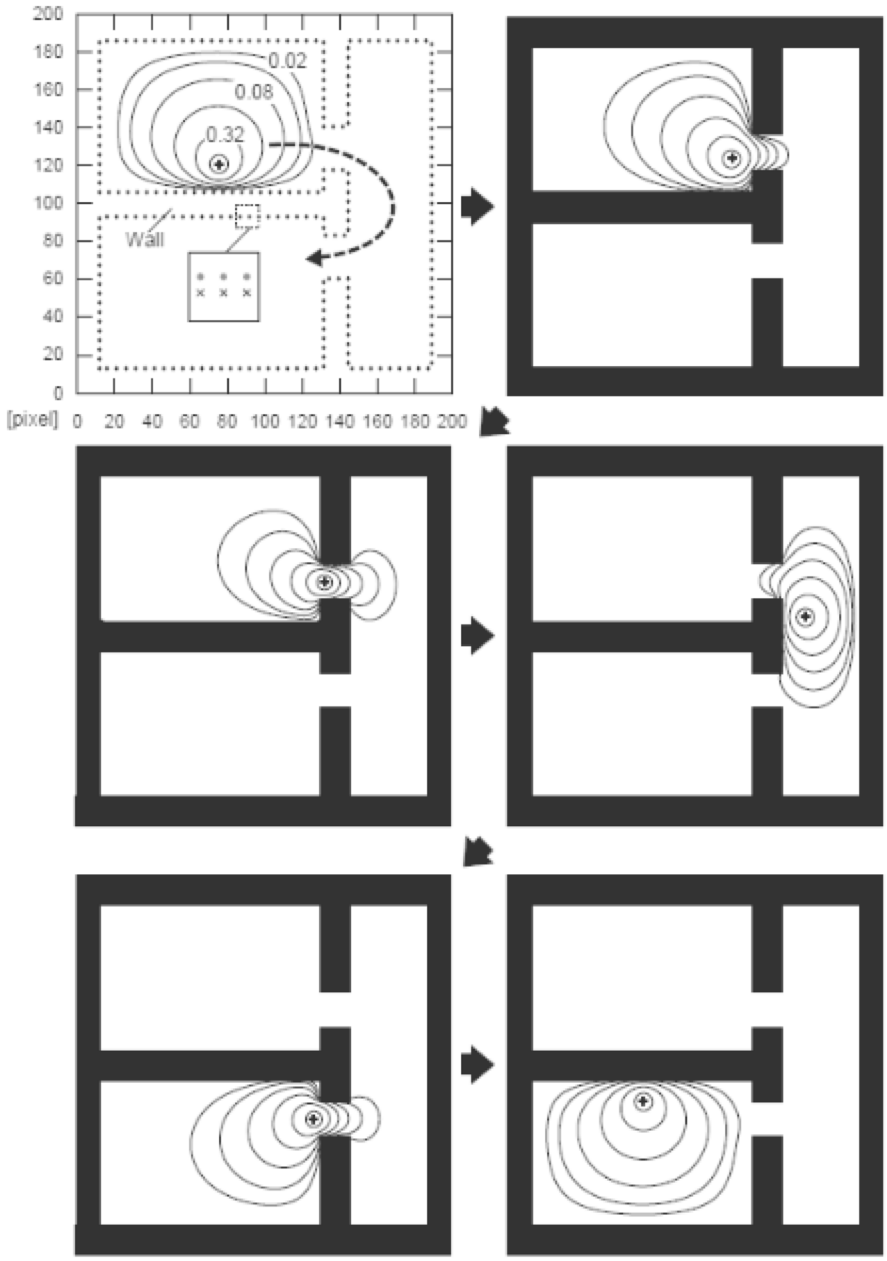


図 4.7: 生成したポテンシャルの例（複雑な例）

3 章で述べた基盤との関係

図 4.8 に，本章で述べる近傍領域生成と 3 章にて述べた基盤との関係を示す．本章で述べる近傍領域は，3 章で述べた基盤上によって提供される仮想世界を対象に生成される．近傍領域生成のための計算は，クライアント（ゲームであればユーザの PC のブラウザ）によって計算することを想定しており，クライアントは一定間隔（多くの場合数秒毎）に基盤から仮想世界の空間データを取得し，そのたびに，得られた空間データの上で注目地点からの近傍領域を生成する．近傍の計算は，セル間の近接相互作用とは関係のないモデルで計算され，計算時にはシミュレーション空間を参照するものの，シミュレーション空間自体の時間発展とは独立して実行される．

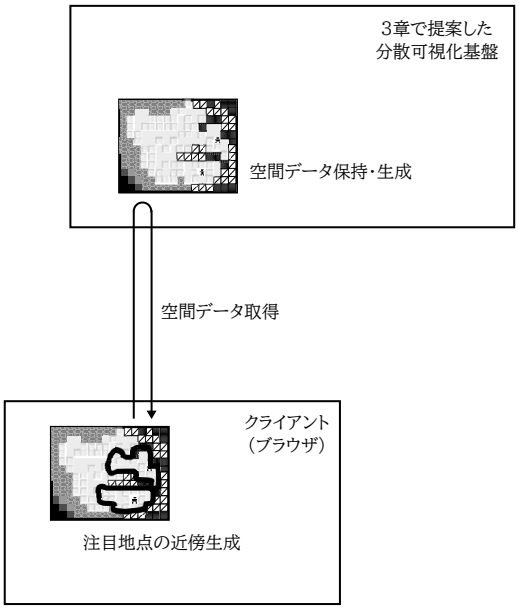


図 4.8: 近傍領域生成と 3 章で述べた基盤との関係

4.5 評価実験

ポテンシャルを生成する注目地点の数を増やした時の計算時間の変動を計測した．計測は，部屋や廊下が存在する建物の中を，複数の人が動き回る状況を想定して実施した．具体的には建屋の壁面に沿って配置した電荷点と標本点の中で，フレーム毎に注目地点（人）の位置を移動させて，式 2.4 を解き，得られた係数に基づいて式 2.3 によってポテンシャルを生成する計算を実施し，計算に要する時間を計測した．検証中のポテンシャルの分布の様子を図 4.9 に示す．また，計算の対象とした建物とそこに配置した電荷点，標本点の仕

様を表 4.1 に，計測結果を図 4.10 に示す．なお，この検証では注目地点同士の近接判定も同時に実行している．近接判定は適当な閾値を定めてそれに基づいてポテンシャル分布の等高線をひき，その等高線の交わりから判定している．

部屋の広さ	横 46m, 縦 34.5m(400x300pixel)
電荷点，標本点の対の数	534
電荷点，標本点の配置間隔	1.15m(10pixel)

表 4.1: 評価に用いた建物構造データの仕様

計測の結果，注目地点の数の増加に応じて計算に要する時間は線形に増加しており，注目地点 1 つあたりのポテンシャルの生成に要する計算時間は約 44 ミリ秒であることがわかった．検証を行った PC の CPU は 2.8GHz（4core）であり，連立方程式の求解には JAMA を利用した．

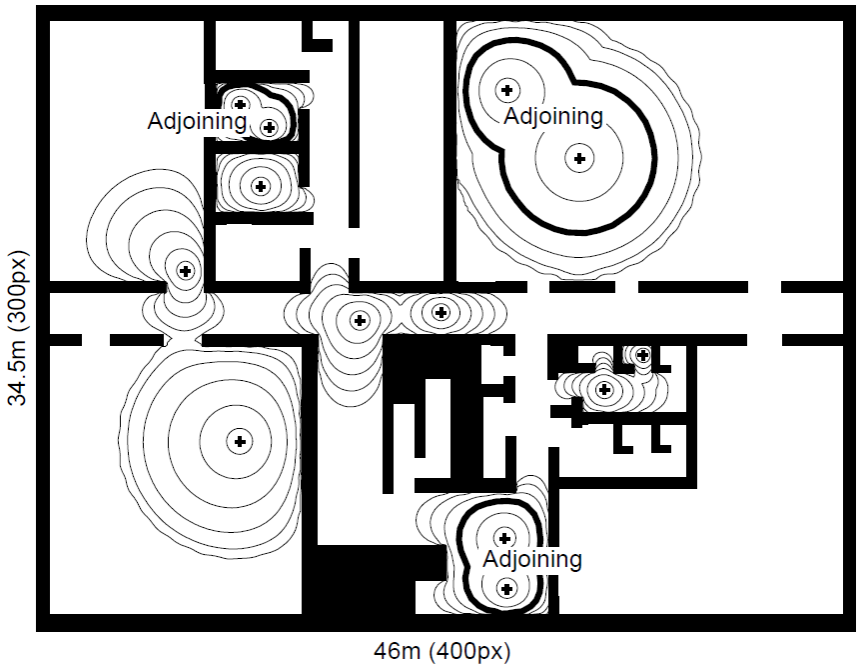


図 4.9: 検証の様子：複数の注目地点に対するポテンシャルを同時に計算．近接判定も併せて実施.（Adjoining が近接と判定された注目地点）

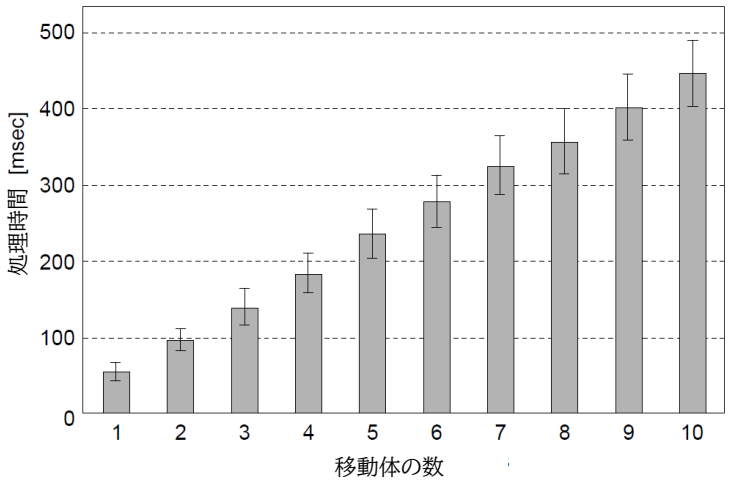


図 4.10: 注目地点の数とポテンシャル生成にかかる時間

4.5.1 誤差について

代用電荷法では，計算時間と生成されるポテンシャル分布の精度はトレードオフの関係にある．配置する電荷点と標本点の数を減らすと連立方程式の元数が減り，計算に要する時間を短縮することができるが，生成されるポテンシャルの誤差もその分大きくなる．図 4.11 に，壁面に沿って配置する電荷点と標本点の間隔と誤差の関係を示す．点の数を多くして配置間隔を短くとった場合は誤差が少なく，点の数を減らして配置間隔を広くとった場合は誤差が多くなっていることがわかる．

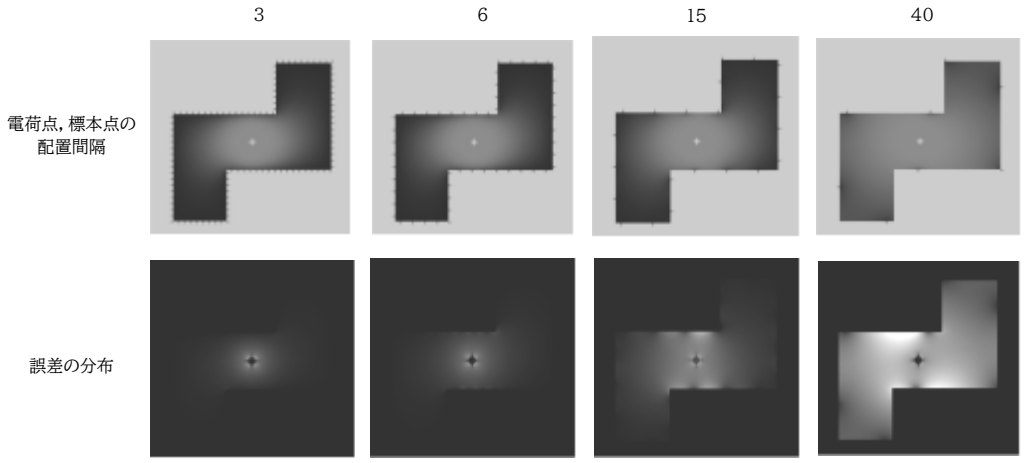


図 4.11: 電荷点，標本点の配置間隔と誤差：上段がポテンシャル分布で下段がその誤差．誤差の量に応じて明るく表示．

実用時の電荷点と標本点の配置間隔は，アプリケーションに応じて，許容できる計算時間と誤差の兼ね合いを考慮して決定することになる．なお，代用電荷法による誤差は電荷点と標本点付近に集中し，結果としてポテンシャルの中心から離れた領域に歪みが生じる．このため，ポテンシャルの中心から限定された距離範囲にある近傍を定義するという目的のためには，多少誤差が大きくても，近傍の範囲にその誤差が含まれないことが多く，実用上支障がない場合が多い．図 4.12 の左に，図 4.6 の右側に示したポテンシャルについて，ポテンシャルの値が低い裾野の領域も含めて細部を図示したものを示す．ポテンシャルの中心から離れた網掛けで示した領域では，ポテンシャルの形状が歪んでいる様子がわかる．これは，例えば図 4.2 で示した状況でポテンシャルを生成した時，図 4.12 の右に示すように，誤差は，近傍として利用する点線で示した領域よりも外側で大きく発生することを示しており，結果として近傍領域内が誤差の影響を大きく受けることが無い．

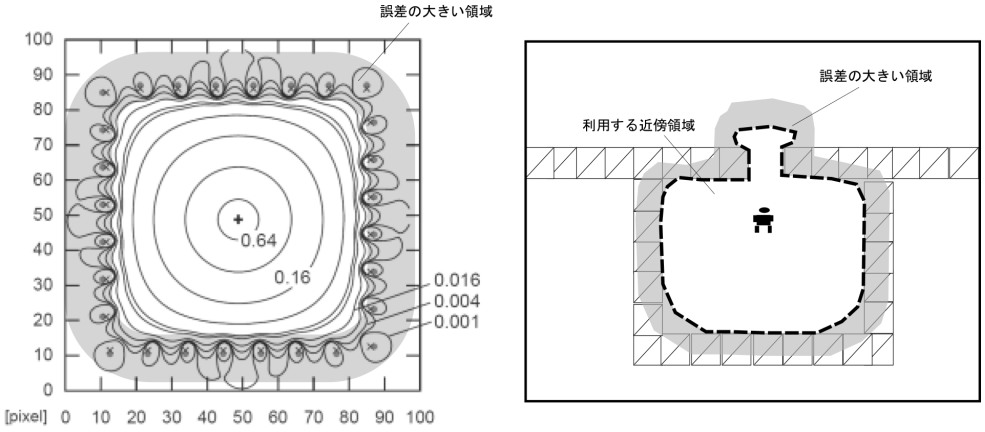


図 4.12: 左 : 代用電荷法によって生成したポテンシャルの細部（ポテンシャルの値が小さい裾野領域も含めて図示．網掛け部分ではポテンシャル形状が歪んでいる）．右 : 図 4.2 の状況下で大きく誤差が発生する領域

4.6 まとめと今後の課題

本章では，3 章で提案した基盤の仮想空間の上で，ユーザがより直感的なインタラクションを行うための，注目地点の近傍領域のモデル化とその高速生成手法について述べた．近傍はポテンシャルによってモデル化し，代用電荷法によってその形状を近似的に求める方式を示した．また，性能計測実験により，注目地点 1 つあたり 50 ミリ秒程度の計算時間

でポテンシャルを生成できることがわかった．ユーザに対して提示される画面のフレームレート要件を 20fps とすると，フレーム毎に許容される処理時間は 50 ミリ秒となるので，注目地点 1 つ分のポテンシャルは画面更新と遅延なく連動して生成できる．また，実験は空間の構造が複雑で電荷点と標本点の数が多い状況で実施した．実際にはこれよりも単純な空間構造で，かつ少ない電荷点と標本点で計算を行うことになる場合が多く，計算時間はさらに短くなることが予想される．

今後の課題として，電荷点と標本点の配置間隔を，アプリケーションの要件と空間構造の複雑さに応じて最適化する方法を提示することが挙げられる．4.5.1 節で示したように，本提案方式によって生成されるポテンシャルの形状の精度は，空間上の構造物に沿って配置する電荷点と標本点の間隔によって異なり，これらが密に多数配置されている方が高い近似精度を得られる．一方で，電荷点と標本点の数は連立方程式の次元数となり，点の数が多い方が計算に要する時間は長くなる．このため，アプリケーションに応じて求められる近傍領域の広さと誤差の兼ね合いを何かしら方式で定式化し，それに応じて電荷点と標本点の配置間隔を最適に決定する方法を検討する．

第5章　組み立て型のストリームデータ処理基盤

5.1　背景

3.1 節で述べたように，例えば大量のコネクテッドカーからアップロードされる緯度・経度や速度などの IoT データを 3 章で提案した基盤（以降，可視化基盤）に入力することで，車両の動態把握や挙動予測をより精度よく行える可能性がある．しかし，アップロードされるセンサデータを直接可視化基盤に送信することは非現実的である．アップロードされるセンサデータの形式はデバイスに依存して異なる場合が多い [131]．例えばコネクテッドカーであれば，車種やメーカーによって，車両に搭載しているセンサや通信装置が異なり，結果としてセンサデータの形式（データに含まれる情報の種類や精度，サンプリング周期やアップロード間隔）が大きく異なる．また，車種やセンサ，通信装置は運用開始後も新しいものが追加されていくため，センサデータの形式は多様かつ変化していくことになる．これらのセンサデータを，可視化基盤に対して直接送信すると，可視化基盤側がセンサデータの多様な解釈処理を実装する必要が生じる．しかし，可視化基盤はその処理を運用中に頻繁に変更することは考慮していないため，新たなセンサデータの形式の追加に対して柔軟に対応することができなくなる．さらに，可視化基盤側でセンサデータの解釈処理を行うと，その解釈のためのロジックが本来のシミュレーションロジックと混在し，結果として構成が複雑化してメンテナンスコスト増大の原因となる．

そこで，クライアントから継続してアップロードされるデータを可視化基盤に直接入力するのではなく，それらのデータをいったん別の基盤で受け付けて，前処理を行った上で可視化基盤に対して適した形式に変換して送信することが必要となる．このように継続的に発生するデータを処理し続けるには，ストリームデータ処理を用いることが適している．図 5.1 に，アップロードされるデータを直接可視化基盤へ入力する場合と，ストリームデータ処理基盤を介して入力する場合の構成を模式的に示す．

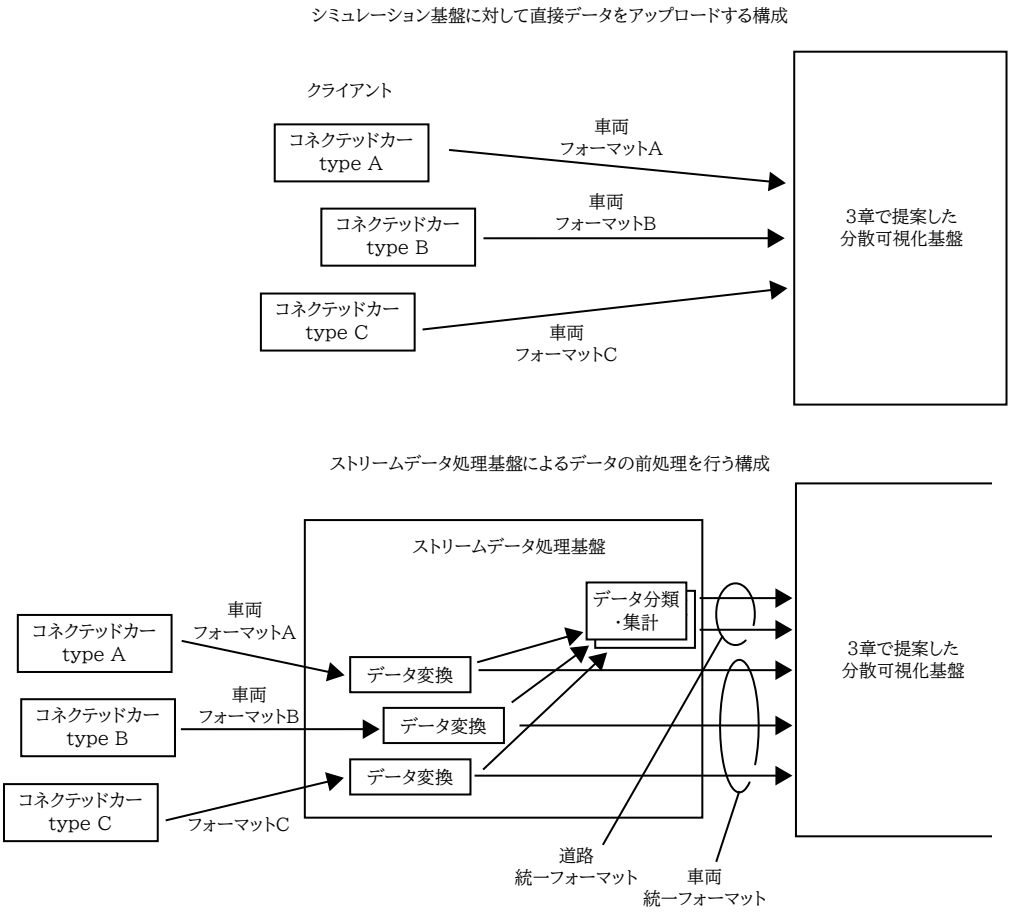


図 5.1: ストリームデータ処理基盤によるデータの前処理.

5.1.1 ストリームデータ処理の課題と本章で述べる研究の目的

本節では IoT データ処理における一般的な要件を挙げ，その上でストリームデータ処理における課題を述べた後に，本章で述べる研究の目的を示す．まず，サービスの開発や運用者から見た IoT データ処理の要件は次の通りである．

1. デバイスとサービスの分離

一般に，IoT デバイスから送信されてくるデータを用いたサービスを開発するには，IoT デバイスの仕様を熟知しておかなければならない．また，デバイスへのアクセス方式や，状況ごとの対処処理を，デバイスの仕様に沿って実装する必要がある．結果として，IoT デバイスからサービスまでが密結合かつ垂直統合的に行われ，一度開発して運用を開始したサービスは，その後の処理内容の変更が難しくなりがちである．また，個々の処理がそのサービスに特化して存在し，処理やデータがサイロ化しがちである．この結合を解くにはデバイスの仕様や状況をサービスから隠蔽し，デバイス

固有の処理をサービスから切り離して実行する必要がある．また，デバイス固有の形式で表現されたデータを一般的な形式に変更する必要がある．

2. サービスの多様性への対応

複数のサービスからの IoT データの利用を可能にするためには，デバイス固有の形式から一般化したデータを，さらにサービスの望む形式に変換し，サービスにとって都合の良いタイミングでサービスに送信する必要がある．

3. 運用中の処理の動的変更

IoT データは継続して発生し続ける一方で，それを利用するサービス側の需要は日々変化していく．また，IoT データの仕様も変化していく．一方，IoT データを利用したサービスは，その時々のリアルタイムな状況に応じて通知や情報提供を行うことが求められるため，データ処理を止めることが望ましくない場合が多い．これに対応するため，サービスの運用中はデータ処理を止めずに処理の内容を変更できる必要がある．

上記の要件に対し，既存のストリームデータ処理基盤は，IoT デバイスからサービスにいたる処理フローを一気通貫で開発することが前提とされており，また，複数の処理フロー同士を並行して連携動作させることを想定していない．結果として，IoT データを複数の処理フローから同時に利用したり，処理フロー同士で情報の交換が行われず，データや処理のサイロ化が生じやすい [126]．また，フロー内の各種の処理が IoT デバイスの仕様に依存したものになりやすい．さらに，運用を開始した後に処理の内容を変更する場合には処理フロー全体を一時的に停止する必要があり，その間にサービスが停止してしまう．本章で述べる研究の目的は，前述した IoT データ処理の要件に応じたストリームデータ処理のこのような課題を解決する枠組みを提案することである．

5.2 関連研究

Apache Flink [4] はステートフルなストリームデータ処理を行う OSS の並列分散ストリームデータ処理基盤であり，分散スナップショット機構による障害発生時の一貫性のあるステート復旧を可能にしている [39].

Flink のストリームデータ処理フローは様々な方式で開発することができる．最も基本的な開発方法は，Flink が提供する DataStream API を用いる方法である．これは，Java および Scala 言語で記述するコードから，Flink 上で処理フローの構築や演算を実行するための

API を適宜呼び出すものである．API を直接利用する方式ではなく，DSL（Domain Specific Language）や抽象化されたプログラミングモデルにもとづいて処理フローを開発する方式も存在している．Apache Beam [3] は，ストリームデータ処理を記述するための抽象化されたプログラミングモデルであり，記述した処理を Flink や Spark，Dataflow [31] などの異なる並列分散ストリームデータ処理基盤によって動作する処理フローに自動変換し実行できる．また，定型化された処理を組み合わせることで処理フローを開発する方式も存在している．Mahapatra [82] は並列分散ストリームデータ処理のフローを構成する要素をコンポーネントとして再利用可能にし，複数のコンポーネントを GUI によって接続することでサービスを容易に構成するマッシュアップツールを提唱している．Ranganathan [94] も，同様に抽象化された処理の単位をコンポーネント化し，それを組み合わせることでストリームデータ処理を記述する枠組みを提唱している．

これらの開発方式はいずれも並列分散ストリームデータ処理基盤上での処理を効率的に記述したり，実行を最適化する上で有効な方式である．しかし，どの方式も，開発した処理フローをストリームデータ処理基盤上で実行した後に，実行し続けながらそのフローに対して変更を施すことが想定されておらず，フローの構造やその中で実行される演算の内容を動的に更新していくことができない．また，定型化された処理を組み合わせる開発方式は，処理フローの構成要素が明確化するため，各要素の粒度や役割を適切に設計することで，前述した密結合・サイロ化の問題を解決できる可能性がある一方で，処理の自由度が低くなり簡易的な処理しか実行できなくなる．これに対し，API を直接利用する方式は自由度が高く複雑な処理フローを開発できる一方で，デバイスからアップロードされたセンサデータの変換や加工・分析の処理フローが垂直統合かつ密結合的に行われサイロ化を引き起こしがちである．

5.3 提案するストリームデータ処理基盤

5.3.1 提案するストリームデータ処理基盤のコンセプト

5.1 節で述べた課題に対応する分散並列ストリームデータ処理基盤（以降，本基盤）を提案する．図 5.2 に本基盤の概要を示す．本基盤の主な特徴は

1. オブジェクトによる実世界の仮想化表現
2. プラグインによる処理の追加・拡張
3. 運用中の動的な処理変更

であり，これは 5.1 節で挙げた課題 1 ～ 3 にそれぞれ対応している．

全体的な構成としては，後述する「オブジェクト」を組み合わせることでサービスを記述する開発環境と，それを分散データストリーム処理基盤のタスクとして展開して配備・実行する実行部から成る．実行部は Flink をベースとしている．

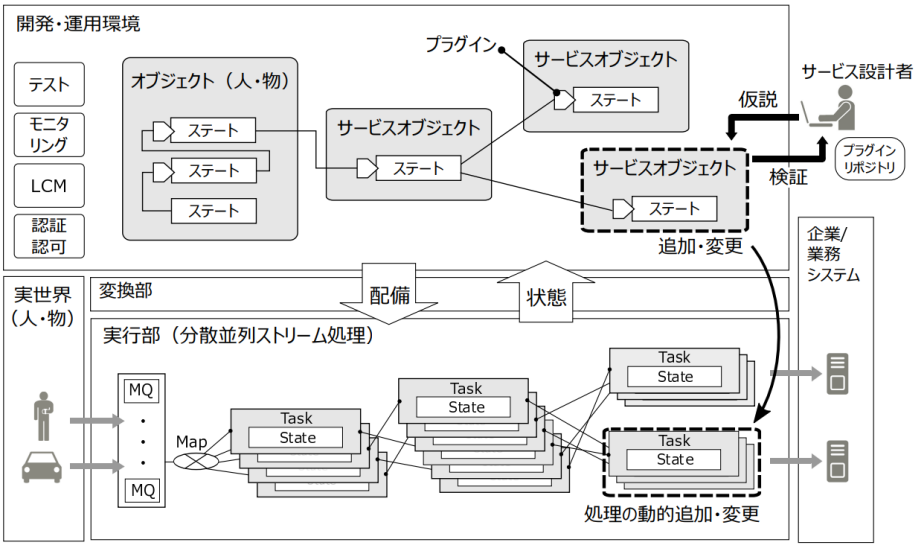


図 5.2: 提案する並列分散ストリームデータ処理基盤の概要

オブジェクトによる実世界の対象物の仮想表現

本基盤では，高頻度で生成される IoT データストリームから，実世界の状況を人や物の単位でオブジェクトとして保持・管理する．オブジェクトは内部にステートを持ち，実世界の人や物の状況と同期してリアルタイムに更新される．また，オブジェクトはデータ処理を追加するためのプラグイン機構を有する．プラグインは，ステートを読み込み，何かしらの処理を行った結果をステートとして書き込むか，あるいは基盤の外へ出力する．

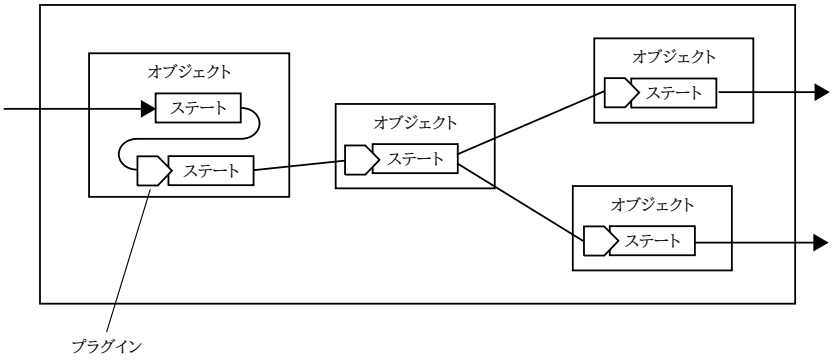


図 5.3: オブジェクトとプラグイン

オブジェクトには「実世界オブジェクト」と「サービスオブジェクト」の 2 種類が存在する．実世界オブジェクトは，実世界の IoT デバイスで 1 対 1 で対応付けられるものであり，サービスオブジェクトは実世界オブジェクトの情報をサービス固有の観点で集約，分析，加工するものである．実世界オブジェクトのステートは「デバイス固有ステート」「一般ステート」「サービス固有ステート」の 3 種類に分けられる（図 5.4）．デバイス固有ステートは IoT デバイスからデータが到着する度に更新されるものであり，デバイスからのデータが未加工に近い形式で保存される．一般ステートは必ずしもサービスの開発者から扱いやすい形式となっていないデバイス固有ステートを一般化・抽象化し，サービス開発者によって理解しやすい形式に変換したものである．ただしこれは特定のサービスに依存した形式ではなく，あくまで複数のサービスから利用しやすい共通形式である．サービス固有ステートは一般ステートを，個々のサービスにとって扱いやすい粒度や通知頻度に変換したものである．各ステート間の変換はプラグインによって行われ，一般ステートを出力するプラグインは IoT デバイスの仕様に詳しい専門家が，またサービス固有ステートを出力するプラグインはサービスの開発者が開発することを想定している．このような 3 階層構成によって IoT デバイスとサービスを疎結合にすることで，サービスがデバイス独自のデータ形式や更新頻度の影響を受けることを抑制する．

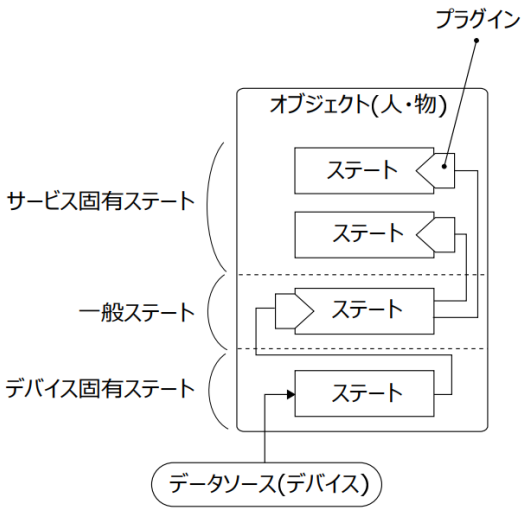


図 5.4: ステートの階層構造

一般的なストリームデータ処理基盤が，サービス要件に応じて設計・実装した処理フローを，運用開始後は変更せずに使い続けることを想定しているのに対し，本基盤はオブジェクトとプラグイン機構によって，サービスの運用開始後も，変化する要件に応じてフローの流れを発展的に変えていく使い方を想定している．

5.3.2 提案するストリームデータ処理基盤の実装方式

本節では，前節で述べたコンセプトを実現するための方式について述べる．そのためにまず，Flink が備えるストリームデータに対する低レベル処理関数である ProcessFunction とキー付けされた状態（Keyed State）に関して述べる．

ProcessFunction と キー付けされた状態（Keyed State）

ProcessFunction[1] は Flink が提供する低レベルのデータストリーム処理関数である．ProcessFunction は 2.3.2 節で示した KeyBy Operator によって生成した Keyed Stream に対して適用することができ，入力されたデータに対する処理を自由に記述することができる．ソースコード 5.1 に ProcessFunction の簡単な実装例を示す．データが到着する度に 6 行目の processElement メソッドが実行され，このメソッドの中でデータの加工や変換処理を自由に記述できる．ここでは入力された文字列の前後に括弧を付与して出力する例を示している．

[1] https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/dev/datastream/operators/process_function/

Listing 5.1: ProcessFunction サンプルコード

```
1  class MyProcessFunction extends
2                              ProcessFunction<String,String>{
3
4    // データが到着する度に呼び出される
5    @Override
6    public void processElement(String in,
7                               Context ctx,
8                               Collector<String> out)
9                               throws Exception {
10
11     // 入力データの前後に括弧を付与する
12     String newValue = "[" + in + "]";
13
14     // 結果を出力する
15     out.collect(newValue);
16   }
17
18 }
```

2.3.1 節で述べたように，ステートフルなストリームデータ処理基盤では Operator が状態値（ステート）を保持できる．この上で，2.3.2 節に示したキー付けされたストリーム（Keyed Stream）に適用した ProcessFunction 内では，ステートをキー付けして（Keyed State），キーごとに分けて読み書きすることができる．この様子を図 5.5 に示す．

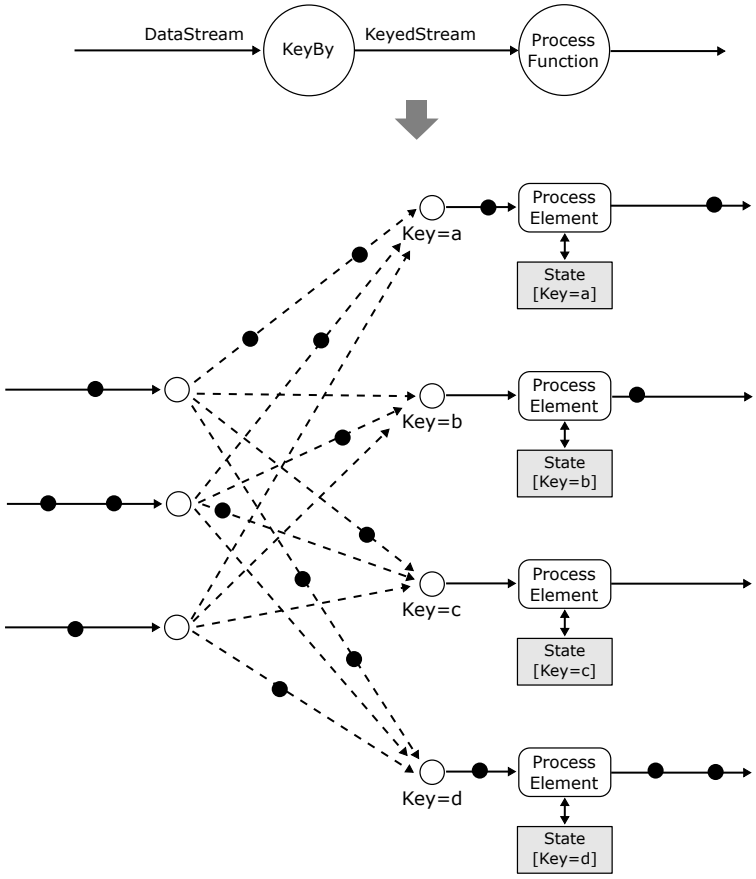


図 5.5: ProcessFunction と Keyed State

Keyed Stream と ProcessFunction によるベース処理フローの生成

本章で提案する基盤では，前節で述べた Keyed Stream と ProcessFunction によって分散ストリームデータ処理の処理フローを事前生成する．一般的に，ストリームデータ処理のフローは，入力されたデータストリームに対して 2.3.2 節で演算を多段に作用させて設計する [10]．これに対し，本提案方式は KeyBy Operator と ProcessFunction のみを多段に繋げてベースとなる処理フローを構成しておき，そのベース処理フローの上に，サービスに応じた処理を定義する方式をとる．5.3.1 節で述べたオブジェクトのプラグインは ProcessFunction の processElemet として，また，オブジェクトステートを Keyed State として実装され，KeyBy Operator による Keyed Stream を介してオブジェクト間の通信が行われる．つまり提案基盤では，Flink が提供するストリームデータ処理用の低レベル API を用いて構築した Keyed Stream のネットワークレイヤの上に，抽象化したオブジェクトレイヤを設けている．この構造を図 5.6 に示す．

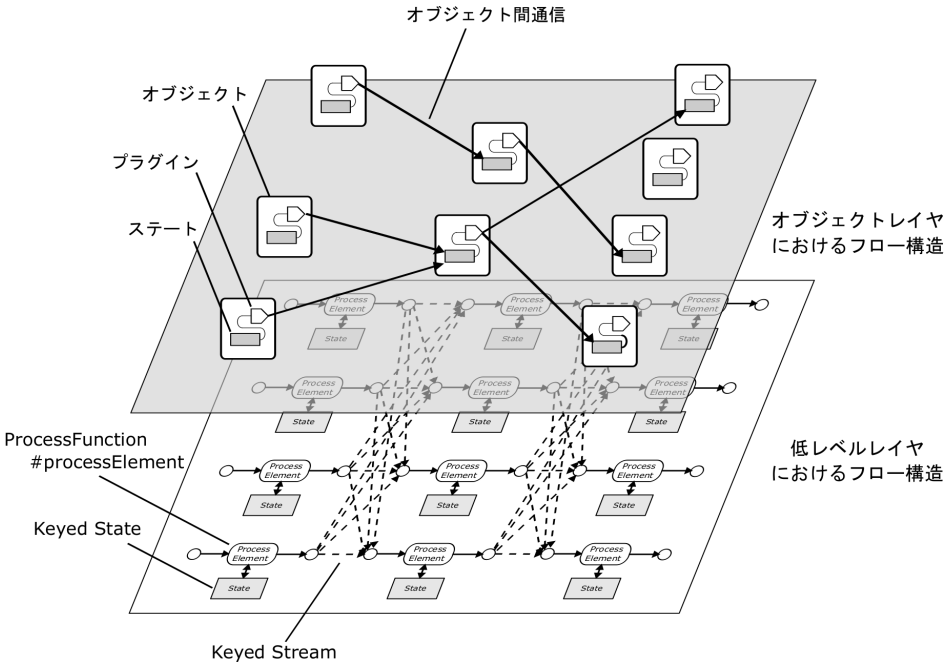


図 5.6: 本章で提案するストリームデータ処理基盤のレイヤ構造

オブジェクトレイヤでは，オブジェクトに到着したデータに対する変換や加工，またその結果を他のオブジェクトに送信する処理をプラグインとして記述するだけで処理フローを構築できる．つまり本基盤では，処理フローの開発者はオブジェクトレイヤのみを意識すればよい．

メッセージベースの処理の動的更新

ProcessEelement において実行する処理は，処理を記述したプログラムを，処理の変更を指示するコマンドメッセージのペイロードに載せて，センサデータと同様に分散ノードに送信することで，変更指示を分散ノードに行き渡らせる．それを受けた ProcessFunction では，ペイロードに載せたプログラムを読み込んでおき，データ到着の度に実行することで，基盤運用中の無停止での処理追加・更新を行う．プラグインの更新機構の概要を図 5.7 に示す．

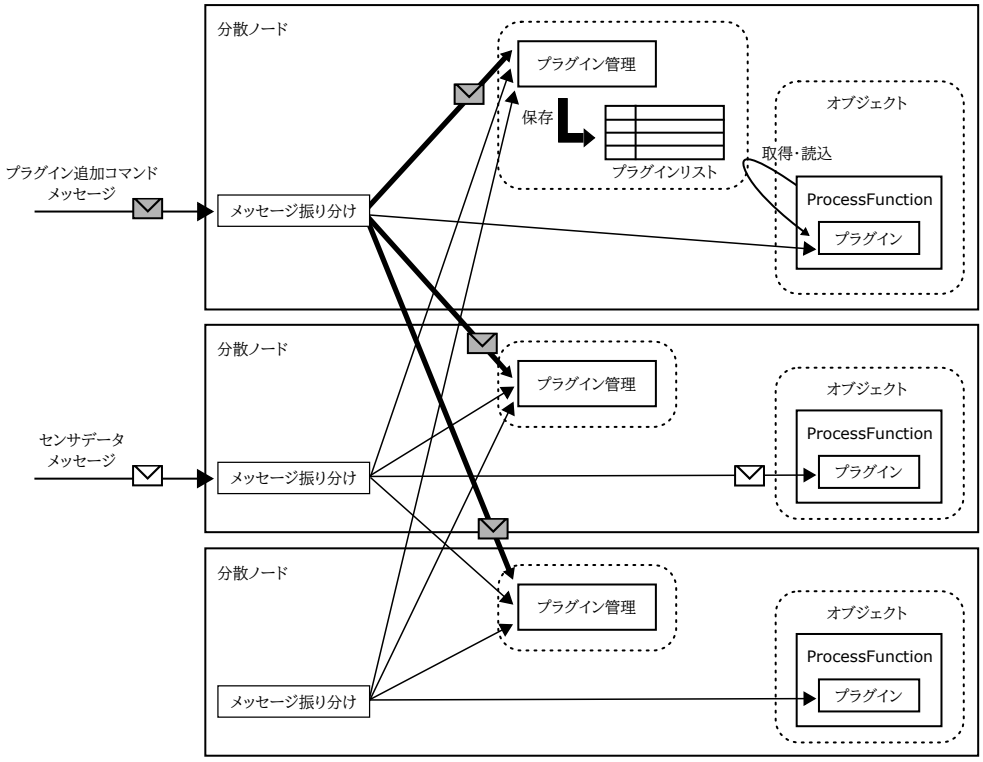


図 5.7: プラグインの動的な更新機構

また，プラグイン更新時は処理の一貫性を保つ仕組みも重要となる．処理の一貫性とは，分散ノードに同じタイミングかつ同じタイムスタンプで到着するデータの処理を行うにあたって，あるデータは変更前のロジックによって，また，あるデータは変更後のロジックによって処理が行われるようなことが生じないことである．このため，本章で提案する基盤では，プラグインの変更を実施する時刻（利用者が付与）を保持し，また，変更前と変更後のプラグインの両方を保持した上で，基盤に到着したデータのタイムスタンプと，プラグイン変更時刻を比較し，それに応じて変更前，変更後のプラグインのどちらを実行するか切替えることで，一貫性を保っている．

5.4 処理フローの試作と性能評価

5.4.1 ワークロードと測定項目

本節では提案基盤でコネクテッドカーを想定した処理フローを試作して実行し，その性能検証を行った結果を述べる．今回の検証では，大量のコネクテッドカーからのデータアップロードを想定し，交通シミュレータである SUMO(Simulation of Urban MObility)[2]によっ

[2]https://www.eclipse.org/sumo/

て生成した，東京都内 142km^2 を走行する，車両 ID，走行速度，緯度経度，走行中の道路 ID などが含まれた車両別の走行データを図 5.8 に示したフローに対して毎秒最大 10 万回投入した（最大で 10 万台の車両が毎秒データをアップロードする想定）．

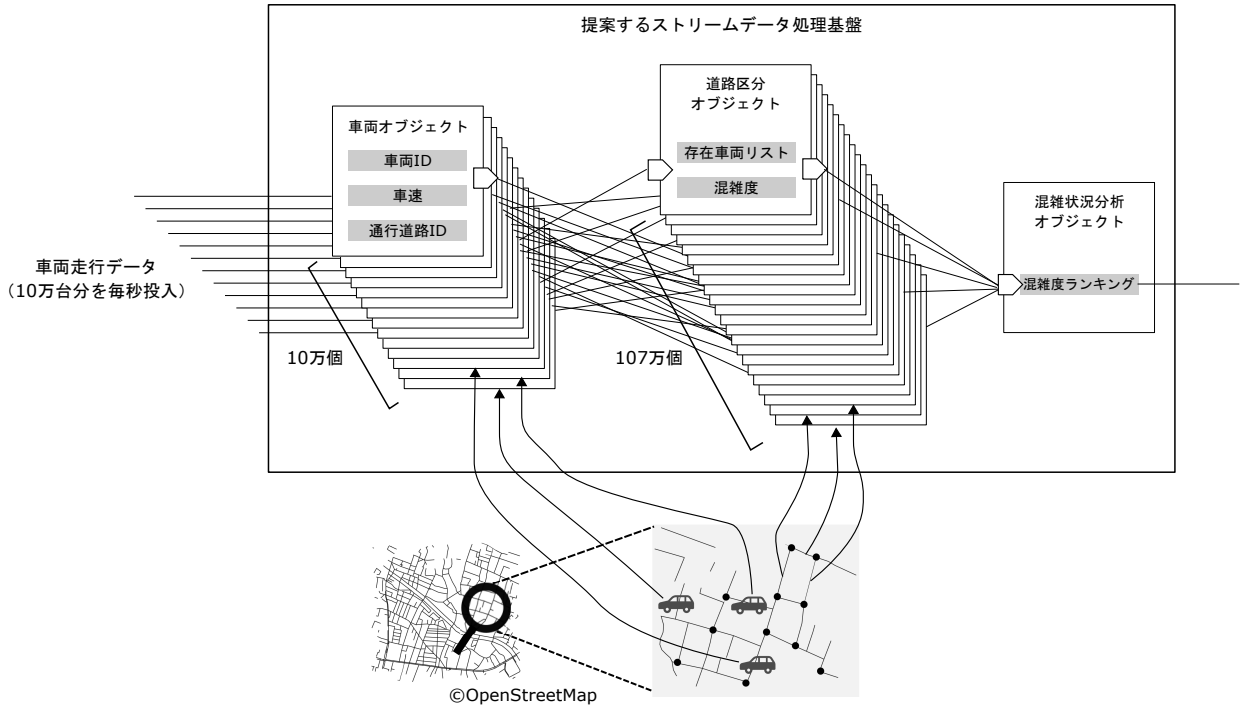


図 5.8: 性能検証に用いた処理フロー

この処理フローは，1 台 1 台の車両を表す「車両オブジェクト」，道路ネットワークを交差点などの結節点ごとに区切った「道路区分オブジェクト」，道路区分の情報を集約する「混雑状況分析オブジェクト」から構成される．全ての車両オブジェクトは走行データを毎秒受信し，単位変換や整形等を行った上で自らのステートとして保存する．また，自らが走行している道路に対して，速度などの情報を道路オブジェクトに対して逐次送信する．道路オブジェクトは，車両オブジェクトから送信されてくる情報にもとづき，車両の存在台数と平均速度からその道路区分の混雑度を計算しステートとして保存する．また，その情報を混雑状況分析オブジェクトへ送信する．道路状況は道路区分オブジェクトから送信されてくるデータを集約し，混雑している箇所のランキングといった混雑状況を分析し，ステートとして保存する．

測定シナリオとして，車両の台数を 1 万台，5 万台，10 万台と増やしていき，それぞれ実行した時の，基盤を構成するホストのリソース消費量を計測した．計測項目は CPU 利用

率，メモリ利用量，ネットワーク転送レートである． 車両オブジェクトはシナリオに応じて 1 万個，5 万個,10 万個を生成し，道路区分オブジェクトは 107 万個を生成した．車両オブジェクトに毎秒送信されるデータのサイズは約 147 キロバイトである．

検証環境

検証のために構築したクラスタ構成は，Flink ノード 4 台，HDFS ノード 3 台，Kafka ノード 1 台である．2.3.1 節で述べたように，HDFS は Flink のステートの永続化のために利用し，今回の検証では，各オブジェクトの Keyed State の内容が定期的に保存される．また Kafka ノードは Flink クラスタに対するデータ投入に利用した．図 5.9 に性能検証時の環境を図示する．全てのノードは同一スペックのマシンを用いた．ノードのスペックを表 5.1 に示す．

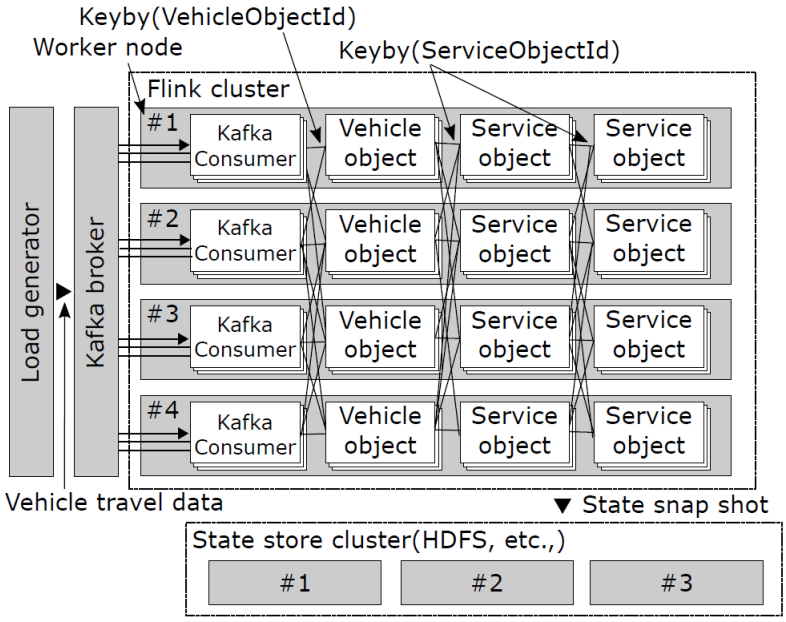


図 5.9: 性能検証時のクラスタ構成

CPU	Intel Xeon Gold 2.40Ghz[20core]
メモリ	195GB
ストレージ	HDD:270GB, SSD:3.6TB
OS	CentOS 7.4.1708

表 5.1: 性能測定に利用したノードのスペック

計測結果と考察

図 5.10 に，投入データ量増加に伴う Flink ノードのリソース消費変化を示す．測定結果ら，投入するデータ量（車両台数）に応じて，各種リソースの消費が線形に増加することが確認できた．また，毎秒 10 万個のデータを投入した時の，各ノードの CPU 利用率は 20% 以下であった．

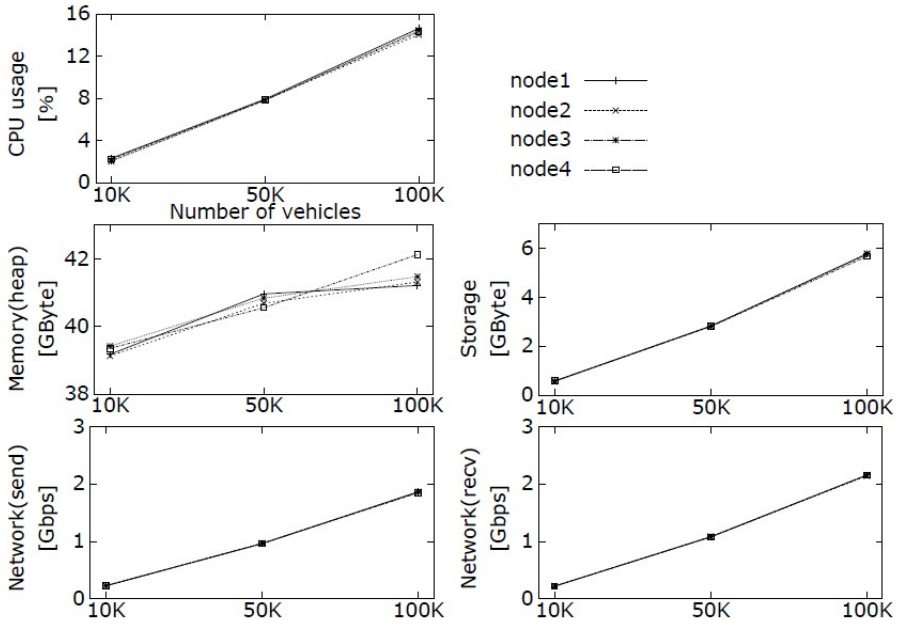


図 5.10: 投入データ量増加に伴うリソース消費変化

また，４つの Flink ノードの負荷が均等に分散されていることが確認できた．これは，今回の処理フローにおいて負荷の多くを占めることになる，1〜10 万個の車両オブジェクトと 107 万個の道路区分オブジェクトの実体である ProcessFunction のサブタスクが，各オブジェクトの ID をもとにしたハッシュ値にもとづいて，各ノードに均等に分配されることに

よる．

5.5 まとめと今後の課題

大量かつ継続的に発生するストリームデータ処理を行う基盤を提案し，サービスを試作した上で性能検証を行った．提案基盤は分散ストリームデータ処理基盤である Flink をベースとし，データフローの柔軟な組み換えを可能にするオブジェクトとプラグイン機構を特徴とする．

5.5.1 IoT データ処理開発における課題に対する本提案方式の貢献

5.1.1 節で示した IoT データ処理における一般的な課題に対する，本章で提案した基盤上での処理フロー開発方式の貢献を次に示す．

IoT データの仕様をサービス開発者が意識不要

サービスの開発者は，IoT データの詳細を意識する必要なく，プラグインによって一般的な形式に変換されたオブジェクトのステートみを把握しておけば，そこからサービスを組み立て可能である．例えば今回試作した処理フローと同様の処理を実運用する場合，車両オブジェクトに入力されるデータは車種やメーカごとに異なる形式となる可能性が高いが，このような場合でも車両オブジェクトのプラグインによってその差異を吸収する処理を実行すれば，それを利用するサービスオブジェクトの開発者は，車種やメーカごとの形式の違いを意識する必要がなくなり，結果としてデバイスとサービスが疎結合になる．これは 5.1.1 節で述べた課題 1 に対する効果である．

IoT データを多様なサービスから利用可能

実世界オブジェクトに対して複数のサービスオブジェクトが，それぞれ固有のプラグインを追加することで，サービスにとって都合の良い形式でデータを受け取ることができる．これによって，IoT データを 1 つのサービスではなく，複数のサービスから利用できるようになる．今回の試作では，コネクテッドカーから生成されるデータを車両オブジェクトのステートとして保持した上で，混雑状況を求めることを目的として道路オブジェクトのみが利用した．しかし，他の観点で，例えば車両ごとに急ブレーキをかけたかどうかを検出するプラグインを車両オブジェクトに追加することで，それをもとに，急制動が多発しているような危険なエリアをリアルタイムに求め

るようなサービスを容易に追加していける．これは 5.1.1 節で述べた課題 2 に対する効果である．

サービスの内容を日々改善していくことができる

プラグインの動的な更新機構によって，データ処理を停止することなく処理フローの一部を更新できる．このため，常に変化し続けるサービスユーザのニーズやビジネス要件に応じて，サービスの内容を改善していくことができる．これは 5.1.1 節で述べた課題 3 に対する効果である．

5.5.2 既存のストリームデータ処理の開発方式に対する位置付け

5.2 節で述べたように，既存のストリームデータ処理の開発方式には幾つかの方式がある．基本的に，開発の容易性と実現できる処理の柔軟性や自由度はトレードオフの関係にある．API を直接利用する開発方式は，自由度が高く複雑な処理フローを開発できる一方で，ストリームデータ処理のプログラミングや動作モデルに関する十分な知見が無いと，性能を十分に引き出す処理フローを開発することが難しい．この対極にある，コンポーネント化された定型処理を組み合わせる方式は，容易に処理フローを作成できる一方で，処理の自由度が低く，複雑な処理を実行するフローを開発できない．

図 5.11 に，5.2 節で述べた従来の処理のフロー開発と，本章で提案する基盤における開発方式を，開発の容易性と処理の自由度の観点で整理した．提案方式は，Keyed Stream と ProcessFunction からなるベース処理フローを下位レイヤに構成した上に抽象化されたオブジェクトレイヤを置き，オブジェクトレイヤのみを開発者に意識させることで，下位のストリームデータ処理のフロー構造を制御する API を直接利用する方式よりも容易な処理フロー開発を可能にしている．一方で，データに対する変換や加工を自由に記述できる ProcessFunction の processElement の実装は，プラグインのコードして自由に記述することができるため，DSL やデータフロー言語などの方式よりも自由度高く処理フローを記述できる．

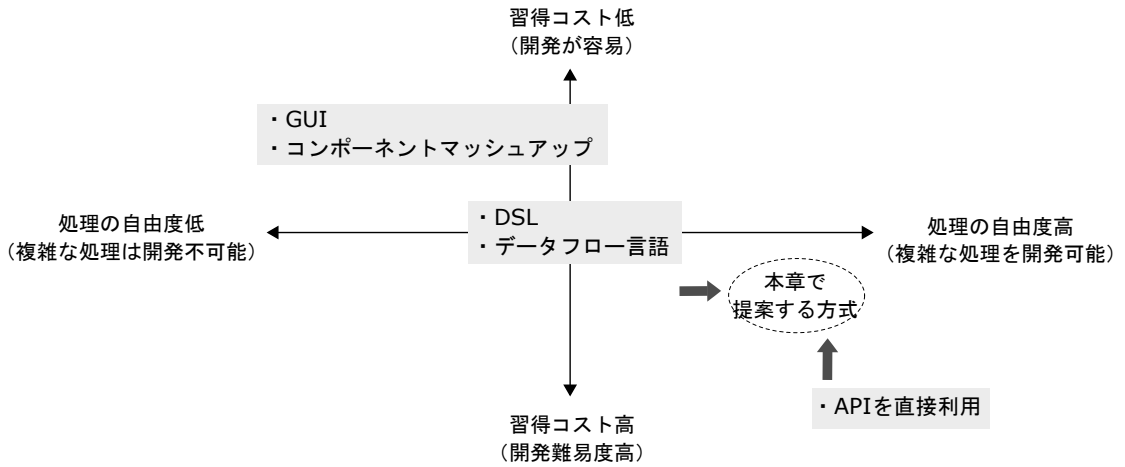


図 5.11: 既存のストリームデータ処理の開発方式と提案方式

今後の課題として，処理負荷の偏りの動的な分散が挙げられる．本提案方式は，実世界に存在する大量のモノをオブジェクトとして表現して扱い，各オブジェクトを複数の計算ノードに均等に割り当てた上でステートの保持や計算を分散する構成をとる．この時，オブジェクトのステートや処理の粒度が大きい場合や，あるいはそのオブジェクトに対して送信されるデータが多い場合は，ステートや処理が十分に分散されず，結果として特定のノードのリソース消費量が偏って高くなる可能性がある．オブジェクトの設計を工夫し，オブジェクトの処理やデータを複数のオブジェクトによって分担する構成をとればこの問題は回避できるが，これを適切に行うにはオブジェクトレイヤの下位に位置するストリームデータ処理や，動作させているマシンのリソースや並列度設定などの情報が必要となる．しかし，本提案方式は，このような下位レイヤに関する知識や情報なしに，容易に処理フローを開発できることを目的としているため，負荷の状況に応じて自動的にオブジェクトのステートやプラグインの処理を分割・複製し，複数のノードに均等に割り付けて負荷を適応的に分散する方式を検討する．

第6章　本研究のまとめ

本論文では，大規模な空間を広域に可視化することを可能とする基盤と，その基盤が保持する空間を対象として，注目地点の近傍領域を生成する方法，および，その基盤に対して入力する IoT データの前処理を行う基盤について述べた．本章ではまず各研究の成果を述べた上で，今後の展望と課題について記す．

6.1　研究の成果

まず，大規模な空間の広域可視化を可能にするための課題を述べた上で，それらを解決するための基盤技術を提案し，実装・評価した．提案した基盤はクラウド上の仮想マシン上で動作するノードを格子状に仮想配置し，ネットワーク接続する構成の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．評価では，2^9 個の格子分（1 格子のサイズ 2 バイト）の空間データをそれぞれ保持するノード（vCPU:4，メモリ:4GiB）を 4 台から 100 台まで接続し，この台数の範囲内ではノード数が増えても各ノードのリソース消費量は一定以内に収まることを確認した．また，クライアントからのリクエスト増加に対するネットワーク送受信量の変動を，従来方式と提案方式とで，理論と計測の両方によって比較し，単位時間あたりのリクエスト数が 40 前後を超えると提案方式のほうが従来方式よりもネットワーク送受信量が少なくなることを確認した．また，リクエスト数の増加に伴う，クライアントに対するデータ提供までのレスポンスも計測し，合計 9 ノードで動作するシステムに対して，70 クライアントからの同時アクセス発生時のレスポンス時間が 20ms 以内（99 パーセンタイル）であることを確認した．また，ノードネットワーク上をデータがマルチホップで伝播していく時間を計測し，少なくとも 10^2 台で構成されるノードネットワーク上では平均するとホップ数の半分程度の秒数でデータが到着することを確認した．さらに，システムを構成するノード群が同一の AZ に存在しない状況下でも，この傾向は変化しないことを併せて確認した．

次に，仮想空間上での注目地点の近傍領域のモデルと高速計算方法を提案した．近傍領域はポテンシャルによって表現し，ラプラス方程式の境界値問題を代用電荷法を応用して近似的に解くことで，高速に近傍領域の分布を計算する方法を示した．また，ポテンシャ

ルの境界条件を低い値に拘束することで，空間上の障害物に近傍分布レベルが吸収される計算方法を示し，単に経路距離のみではなく「気配」を再現することを提案した．また，実在する屋内を想定した空間上で最大 10 名の歩行者が歩き回る状況のもとで各歩行者に対する近傍領域を生成する評価実験を行い，1 人あたり約 50 ミリ秒で近傍分布を生成できることを示した．

そして，提案した基盤に対して IoT デバイスからアップロードされるセンサデータを入力する時の前処理を行うための枠組みを提案した．ここでは大量かつ継続的に発生する IoT データを処理する上での一般的な課題を述べ，それに対応するための分散ストリームデータ処理基盤を提唱した．この基盤は Keyed Stream と ProcessFunction を繰り返し作用させて構築したベース処理フローの上で，プラグインとステートからなるオブジェクト同士を繋げて処理フローを定義するレイヤを設けることを提唱している．また，これによって垂直統合になりがちであった IoT サービス開発を疎結合にできることを，サービスの試作を通じて検証した．

これらはそれぞれ，1 章の図 1.1 で述べた，大規模な仮想空間を対象として，多人数がコミュニケーションをとるようなゲームや，現実世界から取得できるセンサデータを仮想空間上で重畳表示することで，現実世界の状況を適切に把握するシステムを実現するために必要な技術の（1）～（3）に対応しており，本研究はこのようなシステムの実現に寄与したと考えている．

6.2 これからの展望と課題

6.2.1 IaaS（Infrastructure as a Service）の台頭と参加型 P2P による自律分散システム

2020 年の世界パブリッククラウドサービス市場は，前年比 24.1%増の 3120 億ドル[1]であり，大きな成長が続いている．この中で，ユーザがクラウド上に仮想マシンを起動し，OS を含めたソフトウェアを自由に選定して利用できる IaaS（Infrastructure as a Service）分野の市場規模は前年比 33.9%増の 67 億となっており，クラウド上で常時稼働するサーバを立ち上げ，運用することの時間的，金銭的コストが下がり続けている．

3 章で提案した大規模な空間を扱う基盤の構成として，単一組織によって統制運用されるノード群ではなく，多数の協力者が参加・提供するノードが自律的に P2P で協調動作することで全体を構成し，協力者の数の増加に応じて空間データを徐々に増やしていく，参

[1] IDC Worldwide Semiannual Public Cloud Services Tracker, 2H20

加型 P2P による構成が考えられる．現在，IaaS 上で動作している仮想マシンの数は企業向け，個人向け問わず膨大な量が存在しており，これらの仮想マシンの上で P2P プロセスを動作させ，少しずつリソースを供出し合うことで，単一組織では成し得ない規模まで空間を拡大できると考えている．3 章で述べた基盤は 3.1.2 節で述べた通り，局所性と均質性を重視しており，これは統制の効きにくい参加型の P2P による分散システムをシンプルかつ頑強に保つ上で大きなアドバンテージになると考えている．

6.2.2 IoT デバイスの爆発的増加と都市規模のデジタルツイン

センサから得られるデータをもとに実世界に存在する機器，設備，人や車両といった対象物の対となる物の状態値や振る舞いを仮想空間上に再現するデジタルツインという考えが提唱されている [49, 54]．デジタルツインは当初，発電用タービンや航空機エンジンといった整備保全コストが高い物を対象として，その稼働状態をセンサによって取得して仮想空間上で管理・シミュレーションし，効率的なメンテナンスに結びつけようとする取り組みを指していた [11]．これに対して，昨今は，都市に存在する，より大量・多様な物のデータをもとに，都市を丸ごと仮想化しようとする「都市レベルのデジタルツイン」に関する考えが提唱されている [28, 45]．IoT デバイスの数は 2025 年までに全世界で 241 億台に到達すると予測 [16] されており，これらの IoT デバイスから継続的に発生する大量かつ多様なセンサデータを，リアルタイムに処理することで，実社会の状況を都市の規模でリアルタイムに同期した状況を仮想世界に再現できるようになる．

この時に重要な役割を担うのが，リアルタイムデータを対象としたストリームデータ処理を行うためのデータ利活用プラットフォームである．現在，ストリームデータ処理は，サービスの要件を始めに決定した上で利用するデバイスを選定し，そのデータの解釈を含めた処理フローを一気通貫で設計して実装する [46]．しかし，今後は多種多様な IoT デバイスがサービスから切り離されて共通リソースとして存在し，そこから得られるリアルタイムデータをいつでも自由に利用できるようになるべきと考えている．これによって，多様な IoT デバイスを取り込んだサービスを容易に開発・運用できることに加え，新しいビジネスモデルも創出されると考えている．5 章で述べた分散ストリームデータ処理基盤は，オブジェクトによって IoT データを多様なサービスから利用し易い一般的な形式にすることで，IoT のデータソースとサービスを分離し，データ利活用を促すプラットフォームになり得ると考えている．

6.2.3 仮想世界に出現する新しい自然

大規模な空間データを保持・更新・提供できる基盤，および，大量かつ多様な IoT データを柔軟かつリアルタイムに処理できる基盤が揃うことで，クラウド上にいわば「新しい自然」を出現させることができると考えている．例えば地球規模の面積（1 辺 1 メートル換算で約 5.101×10^8 個の格子）と同等の広さを持つ仮想世界が存在し，なおかつその仮想空間をシミュレーションによって時間発展させつつ，また，実世界から IoT デバイスによって取得した情報や，人からのインタラクティブな働きかけによって更新していくことができれば，これまでにない規模の広さで仮想世界とプレイヤ，実世界のモノの相互ダイナミクスを生むことができる．このダイナミクスは，システムの運用者による恣意的な統制や管理によって生み出されるのではなく，あくまで大量のモノや人（ゲームであればプレイヤ）の，仮想空間を介したやり取りが時間的・空間的に蓄積・集約されることで自然発生的に生み出されるものであり，システムの上に発生した自然とみなすことができる．

仮想空間上で生じるダイナミクスの例として創発現象が考えられる．創発現象とは，ミクロ（局所的）な作用が繰り返された結果としてマクロ（大域的）な時間や空間パターンが出現する現象である．自然界には創発現象によって生まれた時間・空間パターンが数多く存在している．例えばメキシコやソマリアなどの準砂漠地帯では，植物の分布が数十メートル間隔の縞模様の空間パターンを形成することが知られており，これは，植物と水の相互作用の結果として創発的に生じたものと考えられている [81]．また，仮想空間内にテレコネクションのような現象も再現できるかもしれない．テレコネクションとは「遠隔相関」や「遠隔結合」とも呼ばれ，離れた地点で何らかの現象が伴って変化する意味であり，具体的な例としてはインドネシア付近と南太平洋東部で海面の気圧が数か月単位で互い違いに変化する「南方振動」が挙げられる．これに類するような，多様な現象を仮想空間内で再現しつつ，それを詳細表示（ミクロな視点）と広域表示（マクロな視点）をシームレスに行き来しながら可視化すれば，仮想空間内で，様々なスケールで生じる多様な現象を確認することができる．これをゲームに応用すれば，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・検証ツールなどへの応用も期待できる．

6.2.4 今後の取組み（NoOps の実現）

前項で述べたような「新しい自然」は，大規模な空間データと保持と更新，および大量の利用者に対する空間データを行うシステムが，途切れることなく継続的に運用されること

で初めて実現され得る．そこで，今後の取組みは，本論文で提案した基盤について，NoOps 化に取り組む．NoOps とは No Uncomfortable Operations の略であり，システムやシステムを支えるインフラの運用に関する様々な作業を自動化，極小化してトイル（Toil: 自動化できるにもかかわらず手作業で行っている，運用にかかわる労働作業）を削減することである [15]．3 章と 5 章で示したシステムは，いずれも複数のノードによって構成される分散システムであり，システムの構成する要素の数が多くなる．これらのノードを個別に管理していては，管理にかかるトイルが多くなり，これがボトルネックとなってシステムの規模に限界が生じる．このため，NoOps によって管理作業の負担を軽減する必要がある．本論文で提案したシステムについて NoOps を実現するためには,「障害影響の局所化」,「負荷偏りの動的分散」,「自己修復」が必要となる．

障害影響の局所化

3 章で提案した可視化基盤は，基盤を構成する分散ノードが格子状に通信を介して繋がり，自らが生成した可視化用のデータを相互に交換し合うことでマルチホップでノード間を流通させることを特徴とする．もし，格子状に繋がった通信経路の上で，特定のノードに障害が発生すると，そのノードを経由して届けられる筈であった，他のノードから送信されてくるデータも届かなくなるものが生じる．そのため，ノードに障害が発生した場合は，障害ノードを辿る経路以外のルートを経て送信されてくるデータが届くように，障害ノード周辺のノードにおけるデータ交換時の拡散半径を拡げるなどの機構を設け，障害の影響を他のノードが受けにくくする必要がある．また，同時多発的にノードがダウンした場合は，ノード同士の繋がりから分断して孤立するノード群が生じる可能性もある．このような時は，分断したノード群の間で情報の交換が行われなくなり，仮想空間も分断してしまう．そのため，情報流通の断絶が発生した際には，そのことを自動的に検出し，障害ノードを挟んだ対向に存在するノードに応急処置的に接続するなどの機構を設ける必要がある．

負荷偏りの動的分散

5 章で示した分散ストリームデータ処理基盤に対する負荷の動的分散の課題については 5.5 節で述べた．負荷の動的分散は 3 章で述べた可視化基盤でも必要となる．

3 章で述べた可視化基盤をゲームに適用する場合，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，プレイヤが一人も存在しない領域が存在し，アクセスしてきたクライ

アントに対するレスポンス処理負荷は，クライアント数に応じて増加することが考えられる．これに対応するため，可視化基盤のノードを動作させるマシンの性能を，混雑する領域に合わせて設定すると，混雑していない領域では性能過剰となり，運用にかかる費用も大きくなる．このため，アクセス量に応じて混んでいる領域のみを対象として自動的に負荷を分散する必要がある．

この課題に対するアプローチとして，クライアントからのアクセス負荷が高まったノードは，自らが周囲のノードに対して送信する空間データの拡散半径を一時的に大きく設定することが考えられる．拡散半径が大きい空間データは周辺のノード群に対してより広く複製されるため，より多くのノードに空間データが複製されることになる．そして，高負荷ノードが参照リクエストをクライアントから受信した場合には，周辺のノード群にそのリクエストに対する応答処理を均等に依頼することで，応答にかかる負荷を分散できる．ただし，この方式ではデータの書き込み要求に対する負荷の分散には適用できない．

自己修復

自己修復とは，ノードで障害が発生しても，システムが自己判断に基づく対処を実施して，人の手を介さずに自動的にシステムを元の状態に修復することである．5 章で提案した分散ストリームデータ処理基盤は，障害時に外部ストレージに保存したステートのスナップショットをもとに自動復旧する機能を有するため，基本的には自己修復機能を有している．3 章で述べた可視化基盤についても，ノードに障害が発生した際にデータの自動復旧を可能とする方式を検討する必要がある．

記号一覧

l_{sw}　サブワールド長

l_w　ワールド長

a_{sw}　サブワールド面積

a_w　ワールド面積

d_{max}　最大 LOD

d_{min}　最小 LOD

dd_n　LOD 階数

a_{swt}　LOD 切替セル数

h_k　高度を 0 から上昇させていった際に k 回目に LOD が切り替わる視点高度

l_{vp}　ビューポート一辺に含まれるセル数

m_{vp}　ビューポート被覆ノード数

r_K　サブワールドデータ（K）の拡散半径

謝辞

本研究を進めるにあたり，ご指導いただきました筑波大学システム情報系の蔡東生准教授に心から感謝いたします．思えば，３年ほど前に，当時はまったく面識の無かった蔡先生の研究室へ突然訪問し，それ以来，先生の研究室はもちろんのこと，他大学のキャンパス内，喫茶店，リモートなど，社会人である私の事情を考慮し，時と場所を問わず，始終あたたかいご指導を頂きました．そのおかげが今日の研究成果に繋がったと考えています．また，筑波大学計算科学研究センターの天笠俊之教授，高橋大介教授，筑波大学システム情報系の新城靖准教授，坪内孝司教授，三谷純教授には本研究を進める上で重要となるご助言，ご指導を受け賜りました．また，本論文の審査を快く引き受けていただき，深く感謝を申しあげます．

なお，この研究は株式会社富士通研究所（現富士通株式会社）の博士号取得支援によって行われたものです．私の上司でもある同研究所の松井一樹プロジェクトディレクター（現上席研究員），植木美和プロジェクトマネージャー，水谷政美プロジェクトディレクターには，業務の一環として本研究を進めることに，ご理解をいただきました．この場をかりて，お礼を申し上げます．

最後に，これまで私を応援してくれた母と他界した父，また，研究を行うための環境を与えてくれた妻の安紗子，長女の紗也に感謝したいと思います．そして，新しく生まれてこようとしている息子に心から感謝します．

参考文献

[1] Amazon kinesis data streams. https://aws.amazon.com/kinesis/data-streams/.

[2] The apache software foundation: Apache accumulo. https://accumulo.apache.org/.

[3] The apache software foundation: Apache beam. http://beam.apache.org/.

[4] The apache software foundation: Apache flink. http://flink.apache.org/.

[5] The apache software foundation: Apache hadoop. http://hadoop.apache.org/.

[6] The apache software foundation: Apache hbase. https://hbase.apache.org/.

[7] The apache software foundation: Apache kafka. https://kafka.apache.org/.

[8] The apache software foundation: Apache spark. https://spark.apache.org/.

[9] Applying the kappa architecture in the telco industry. https://www.oreilly.com/content/applying-the-kappa-architecture-in-the-telco-industry/.

[10] Flink datastream api programming guide. https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/dev/datastream/overview/.

[11] Ge reports japan デジタルツイン. https://www.gereports.jp/tag/デジタルツイン.

[12] Geoserver: Mongodb data store. https://docs.geoserver.org/latest/en/user/extensions/mongodb/index.html.

[13] Github googlelebeldb. https://github.com/google/leveldb.

[14] Mongodb, inc.: The most popular database for modern apps — mongodb. https://www.mongodb.com/.

[15] Noops definition v1.0. https://github.com/noopsjapan/community/blob/master/DEFINITION.md/.

[16] Number of internet of things (iot) connected devices worldwide from 2019 to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[17] Oracle corporation and/or its affiliates: Mysql. https://www.mysql.com/.

[18] Oracle ’s spatial database. https://www.oracle.com/database/spatial/.

[19] P2p systems: Gossip protocols. https://www.cs.cornell.edu/courses/cs6410/2017fa/slides/20-p2p-gossip.pdf.

[20] A persistent key-value store for fast strage environment. http://rocksdb.org/.

[21] The postgresql global development group: Postgresql: The world’s most advanced open source relational database. https://www.postgresql.org/.

[22] Redis. https://redis.io/.

[23] Spatial and geographic objects for postgresql. http://www.postgis.net/.

[24] Upc consortium: Upc language specifications version 1.3 (november 16, 2013). https://upc.lbl.gov/docs/user/upc-lang-spec-1.3.pdf.

[25] Xcalablemp specification working group: Xcalablemp language specification version 1.4 (2018). https://xcalablemp.org/download/spec/xmp-spec-1.4.pdf.

[26] 東京都デジタルツイン実現プロジェクト. https://info.tokyo-digitaltwin.metro.tokyo.lg.jp/.

[27] 平成 29 年版情報通信白書. https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h29/html/na000000.html.

[28] 都市の「デジタルツイン」の構想と可能性. https://www.accenture.com/jp-ja/insights/public-service/digitaltwin-city, 2020.

[29] Paul C. Adams. Teaching and learning with simcity 2000. *Journal of Geography*, Vol. 97, No. 2, pp. 47–55, 1998.

[30] Sarthak Agarwal and K. Rajan. Analyzing the performance of nosql vs. sql databases for spatial and aggregate queries. 2017.

[31] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, Vol. 8, pp. 1792–1803, 2015.

[32] Mehrabian Albert. *Nonverbal Communicatioin*. 1972.

[33] K. Amano. A charge simulation method for the numerical conformal mapping of interior, exterior and doubly-connected domains. *Journal of Computational and Applied Mathematics*, Vol. 53, No. 3, pp. 353 – 370, 1994.

[34] Yasushi Ando, Yoshiaki Fukazawa, Osamu Masutani, Hirotoshi Iwasaki, and Shinichi Honiden. Performance of pheromone model for predicting traffic congestion. Vol. 2006, pp. 73–80, 01 2006.

[35] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab, 2004.

[36] Uwe Arnold, Heinrich Subke, and Maria Reichelt. Simcity in infrastructure management education. *Education Sciences*, Vol. 9, No. 3, 2019.

[37] David Bar-El and Kathryn E. Ringland. Crafting game-based learning: An analysis of lessons for minecraft education edition. In *International Conference on the Foundations of Digital Games*, FDG '20, New York, NY, USA, 2020. Association for Computing Machinery.

[38] S. Benford and Lennart E. Fahlén. A spatial model of interaction in large virtual environments. In *ECSCW*, 1993.

[39] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. State management in apache flink®: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, Vol. 10, No. 12, p. 1718–1729, August 2017.

[40] Paris Carbone, Asterios Katsifodimos, † Kth, Sics Sweden, Stephan Ewen, Volker Markl, Seif Haridi, Kostas Tzoumas. Apache flink™: Stream and batch processing in a single engine. *IEEE Data Engineering Bulletin*, Vol. 38, , 01 2015.

[41] Sirish Ch, Owen Cooper, Amol Deshp, Michael Franklin, Joseph Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. Telegraphcq: Continuous dataflow processing for an. 12 2002.

[42] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, Vol. 26, No. 2, June 2008.

[43] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Peng, and Paul Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. pp. 1789–1792, 05 2016.

[44] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Dale Woodford, Yasushi Saito, Christopher Taylor, Michal Szymaniak, and Ruth Wang. Spanner: Google's globally-distributed database. In *OSDI*, 2012.

[45] Fabian Dembski, Uwe Wössner, Mike Letzgus, Michael Ruddat, and Claudia Yamu. Urban digital twins for smart cities and citizens: The case study of herrenberg, germany. *Sustainability*, Vol. 12, No. 6, p. 2307, Mar 2020.

[46] Pratikkumar Desai, Amit Sheth, and Pramod Anantharam. Semantic gateway as a service architecture for iot interoperability. 10 2014.

[47] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, Vol. 1, No. 1, p. 269–271, December 1959.

[48] Glenn Ekaputra, Charles Lim, and I Eng Kho. Minecraft: A game as an education and scientific learning tool. 12 2013.

[49] A. El Saddik. Digital twins: The convergence of multimedia technologies. *IEEE MultiMedia*, Vol. 25, No. 2, pp. 87–92, 2018.

[50] Lennart E. Fahlén, Charles Grant Brown, Olov Ståhl, and Christer Carlsson. A space based model for user interaction in shared synthetic environments. In *Proceedings of the*

INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, CHI '93, p. 43–48, New York, NY, USA, 1993. Association for Computing Machinery.

[51] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, Vol. C-21, No. 9, pp. 948–960, 1972.

[52] Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, Vol. 202, pp. 18 – 27, 2017. Big Remotely Sensed Data: tools, applications and experiences.

[53] Chris Greenhalgh and Steven Benford. Massive: A collaborative virtual environment for teleconferencing. *ACM Trans. Comput.-Hum. Interact.*, Vol. 2, No. 3, p. 239–261, September 1995.

[54] Michael Grieves and John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, pp. 85–113. 08 2017.

[55] Qingfeng Guan. prpl: an open-source general-purpose parallel raster processing programming library. *SIGSPATIAL Special*, Vol. 1, pp. 57–62, 01 2009.

[56] Qingfeng Guan and Keith C. Clarke. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science*, Vol. 24, No. 5, pp. 695–722, 2010.

[57] Xuefeng Guan, Chong Xie, Linxu Han, Yumei Zeng, Dannan Shen, and Weiran Xing. Map-vis: A distributed spatio-temporal big data visualization framework based on a multi-dimensional aggregation pyramid model. *Applied Sciences*, Vol. 10, No. 2, 2020.

[58] N. Guo, W. Xiong, Q. Wu, and N. Jing. 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.

[59] Z.J. Haas, J.Y. Halpern, and Li Li. Gossip-based ad hoc routing. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, pp. 1707–1716 vol.3, 2002.

[60] Stefan Hagedorn, Philipp Götze, Omran Saleh, and Kai-Uwe Sattler. Stream processing platforms for analyzing big dynamic data. *it - Information Technology*, Vol. 58, No. 4, pp. 195–205, 2016.

[61] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100–107, 1968.

[62] John Hershberger and Subhash Suri. An optimal algorithm for euclidean shortest paths in the plane. *SIAM J. Comput.*, Vol. 28, pp. 2215–2256, 1999.

[63] Fei Hu, Mengchao Xu, Jingchao Yang, Yanshou Liang, Kejin Cui, Michael M. Little, Christopher S. Lynnes, Daniel Q. Duffy, and Chaowei Yang. Evaluating the open source data containers for handling big geospatial raster data. *ISPRS International Journal of Geo-Information*, Vol. 7, No. 4, 2018.

[64] James N. Hughes, Andrew Annex, Christopher N. Eichelberger, Anthony Fox, Andrew Hulbert, and Michael Ronquest. GeoMesa: a distributed architecture for spatio-temporal fusion. In Matthew F. Pellechia, Kannappan Palaniappan, Peter J. Doucette, Shiloh L. Dockstader, Gunasekaran Seetharaman, and Paul B. Deignan, editors, *Geospatial Informatics, Fusion, and Motion Video Analytics V*, Vol. 9473, pp. 128 – 140. International Society for Optics and Photonics, SPIE, 2015.

[65] Ihara and Mori. Autonomous decentralized computer control systems. *Computer*, Vol. 17, No. 8, pp. 57–66, 1984.

[66] Takuji Iimura, Hiroaki Hazeyama, and Youki Kadobayashi. Distributed scalable multiplayer online game servers on peer-to-peer networks. *IPSJ Digital Courier*, Vol. 1, pp. 75–90, 2005.

[67] Muhammad Iqbal and Tariq Soomro. Big data analysis: Apache storm perspective. *International Journal of Computer Trends and Technology*, Vol. 19, pp. 9–14, 01 2015.

[68] Jared Jardine and Daniel Zappala. A hybrid architecture for massively multiplayer online games. 01 2008.

[69] HyeongYeop Kang, Hanyoung Jang, Chang-Sik Cho, and JungHyun Han. Multi-resolution terrain rendering with gpu tessellation. *The Visual Computer*, Vol. 31, , 04 2014.

[70] M. Katsurada and H. Okamoto. The collocation points of the fundamental solution method for the potential problem. *Computers And Mathematics with Applications*, Vol. 31, No. 1, pp. 123 – 137, 1996.

[71] Masashi Katsurada. Asymptotic error analysis of the charge simulation method in a jordan region with analytic boundary. *Journal of the Faculty of Science. Section I A*, Vol. 37, , 01 1990.

[72] Takuma Kawamura and Yasuhiro Idomura. Improvement in interactive remote in situ visualization using simd-aware function parser and asynchronous data i/o. *Journal of Visualization*, Vol. 23, , 06 2020.

[73] Minsung Kim and Jungyeop Shin. The pedagogical benefits of simcity in urban geography education. *Journal of Geography*, Vol. 115, No. 2, pp. 39–50, 2016.

[74] Jon Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, p. 163–170, New York, NY, USA, 2000. Association for Computing Machinery.

[75] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pp. 1398–1404 vol.2, 1991.

[76] Jay Kreps. Questioning the lambda architecture. `https://www.oreilly.com/radar/questioning-the-lambda-architecture/`.

[77] Margy KUNTZ. *Teacher's Guide, An Educational Companion for SimCity 3000*, 1999.

[78] Avinash Lakshman and Prashant Malik. Cassandra: Structured storage system on a p2p network. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, p. 5, New York, NY, USA, 2009. Association for Computing Machinery.

[79] Dan Li, Xia Li, XiaoPing Liu, YiMin Chen, ShaoYing Li, Kai Liu, JiGang Qiao, YiZhong Zheng, YiHan Zhang, and ChunHua Lao. Gpu-ca model for large-scale land-use change simulation. *Chinese Science Bulletin*, Vol. 57, No. 19, pp. 2442–2452, Jul 2012.

[80] Peter Lindstrom, David Koller, William Ribarsky, Larry Hodges, Augusto Op, and N. Faust. An integrated global gis and visual simulation system. 08 1999.

[81] W. A. Macfadyen. Vegetation patterns in the semi-desert plains of british somaliland. *The Geographical Journal*, Vol. 116, No. 4/6, pp. 199–211, 1950.

[82] Tanmaya Mahapatra, Ilias Gerostathopoulos, Federico Alonso Fernández Moreno, and Christian Prehofer. Designing flink pipelines in iot mashup tools. 2018.

[83] Nathan Marz. *Big data: principles and best practices of scalable realtime data systems*. O'Reilly Media, 2013.

[84] John Minnery and Glen Searle. Toying with the city? using the computer game simcity 4 in planning education. *Planning Practice & Research*, Vol. 29, No. 1, pp. 41–55, 2014.

[85] Steve Nebel, Sascha Schneider, and Gnter Daniel Rey. 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.

[86] Steve Nebel, Sascha Schneider, and Gunter Daniel Rey. 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.

[87] Nils J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *IJCAI*, 1969.

[88] S. Nishimura, S. Das, D. Agrawal, and A. E. Abbadi. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In *2011 IEEE 12th International Conference on Mobile Data Management*, Vol. 1, pp. 7–16, 2011.

[89] Takafumi Onishi, Julius Michaelis, and Yasuhiko Kanemasa. Recovery-conscious adaptive watermark generation for time-order event stream processing. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 66–78, 2020.

[90] Alexandra Overby and Brian L. Jones. Virtual legos: Incorporating minecraft into the art education curriculum. *Art Education*, Vol. 68, No. 1, pp. 21–27, 2015.

[91] Maria Patrou, Mahbub Alam, Puya Memarzia, Suprio Ray, Virendrakumar Bhavsar, Kenneth Kent, and G.W. Dueck. Distil: a distributed in-memory data processing system for location-based services. pp. 496–499, 11 2018.

[92] Yichen Qian. The application and improvement discussion of gossip in NoSQL databases: Taking cassandra as an analysis example. *Journal of Physics: Conference Series*, Vol. 1437, No. 1, p. 012001, jan 2020.

[93] Nikunj Raghuvanshi, Nico Galoppo, and Ming C. Lin. Accelerated wave-based acoustics simulation. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, SPM '08, p. 91–102, New York, NY, USA, 2008. Association for Computing Machinery.

[94] Anand Ranganathan. Supporting location-based services through composable stream-processing flows. In *2010 Eleventh International Conference on Mobile Data Management*, pp. 320–325, 2010.

[95] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5, pp. 501–518, 1992.

[96] Niklas Röber, Ulrich Kaminski, Maic Masuch, and Otto von Guericke-University Magdeburg. Ray acoustics using computer graphics technology. 2007.

[97] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages*, pp. 336–343, 1996.

[98] Fernando Silva-Coira, José R. Paramá, Susana Ladra, Juan R. López, and Gilberto Gutiérrez. Efficient processing of raster and vector data. *PLOS ONE*, Vol. 15, No. 1, p. e0226943, Jan 2020.

[99] Mamoru Takano, Atsushi Kashiwabara, Hiroshi Tamura, Keisuke Nakano, and Masakazu Sengoku. A consideration of method of information sources moving with covering problem for distribution of regional information. *Transaction of the Japan Society for Simulation Technology*, Vol. 11, No. 2, pp. 15–22, 2019.

[100] William Tärneberg, Vishal Chandrasekaran, and Marty Humphrey. Experiences creating a framework for smart traffic control using aws iot. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pp. 63–69, 2016.

[101] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, SIGMOD '92, p. 321–330, New York, NY, USA, 1992. Association for Computing Machinery.

[102] P. Vadakkepat, Kay Chen Tan, and Wang Ming-Liang. Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, Vol. 1, pp. 256–263 vol.1, 2000.

[103] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, 2015. Association for Computing Machinery.

[104] B.D. Vleeschauwer, Bruno Van Den Bossche, T. Verdickt, Filip De Turck, Bart Dhoedt, and Piet Demeester. Dynamic microcell assignment for massively multiplayer online gaming. *4th ACM SIGCOMM workshop on Network and System Support for Games*, pp. 1–7, 01 2005.

[105] Yi Wang, Arnab Nandi, and Gagan Agrawal. Saga: Array storage as a db with support for structural aggregations. 06 2014.

[106] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, Vol. 393, No. 6684, pp. 440–442, Jun 1998.

[107] Stephen Wolfram. *Cellular automata and complexity : collected papers*. Addison-Wesley Pub. Co., 1994.

[108] Chang Xia, Haijun Wang, Anqi Zhang, and Wenting Zhang. A high-performance cellular automata model for urban simulation based on vectorization and parallel computing technology. *International Journal of Geographical Information Science*, Vol. 32, No. 2, pp. 399–424, 2018.

[109] Tsai-Wei Yang and Kuochen Wang. Failure detection service with low mistake rates for sdn controllers. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–6, 2016.

[110] K. Yasumoto and K. Nahrstedt. Ravitas: Realistic voice chat framework for cooperative virtual spaces. In *2005 IEEE International Conference on Multimedia and Expo*, pp. 1046–1049, 2005.

[111] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, New York, NY, USA, 2015. Association for Computing Machinery.

[112] Wang Yuzhi, Zhang Liguo, Sun Jiuhu, and Xiang Hengmao. Research on real-time visualization of massive data oriented to digital earth. 12 2009.

[113] A. Zelinsky, R.A. Jarvis, J. C. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *In Proceedings of International Conference on Advanced Robotics*, pp. 533–538, 1993.

[114] J. Zhang and D. Wang. High-performance zonal histogramming on large-scale geospatial rasters using gpus and gpu-accelerated clusters. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pp. 993–1000, 2014.

[115] Jianting Zhang and Simin You. High-performance quadtree constructions on large-scale geospatial rasters using gpgpu parallel primitives. *International Journal of Geographical Information Science*, Vol. 27, No. 11, pp. 2207–2226, 2013.

[116] Jianting Zhang, Simin You, and Le Gruenwald. Large-scale spatial data processing on gpus and gpu-accelerated clusters. *SIGSPATIAL Special*, Vol. 6, No. 3, p. 27–34, April 2015.

[117] Nikolaos Zygouras, Nikos Zacheilas, Vana Kalogeraki, Dermot Kinane, and Dimitrios Gunopulos. Insights on a scalable and dynamic traffic management system. 03 2015.

[118] 靖志安藤, 良彰深澤, 修増谷, 宏佐々木, 弘利岩崎, 真一本位田. フェロモンモデル : 交通渋滞予測への適用. 電子情報通信学会論文誌. D-I, 情報・システム, I-情報処理 = The

transactions of the Institute of Electronics, Information and Communication Engineers. D-I, Vol. 88, No. 9, pp. 1287–1298, sep 2005.

[119] 澄彦夏, 芳樹佐藤, 滋千葉. 軽量で抽象度の高い条件付きバリア同期とその実装方法. 情報処理学会論文誌プログラミング（PRO）, Vol. 8, No. 3, pp. 11–21, sep 2015.

[120] 俊哉吉良, 浩平床井. ポテンシャル法による多数アバターの自律的行動生成 (キャラクタ・セキュリティ(ポスター), 映像表現・芸術科学フォーラム 2015). 映像情報メディア学会技術報告, Vol. 39.14, No. 0, pp. 123–125, 2015.

[121] 俊哉吉良, 浩平床井. ポテンシャル法による地形戦術を活用した大規模歩兵戦闘の自律的行動生成. 画像電子学会研究会講演予稿, Vol. 15.04, No. 0, pp. 41–44, 2016.

[122] 森欣司, 宮本捷二, 井原廣一. 自律分散概念の提案. 電気学会論文誌. C, Vol. 104, No. 12, pp. 303–310, 1984.

[123] 経済産業省, KPMG コンサルティング. 【報告書】令和２年度コンテンツ海外展開促進事業（仮想空間の今後の可能性と諸課題に関する調査分析事業）. Technical report, 2021.

[124] 建部修見, 智嗣関口. 共有メモリ計算機における局所同期機構. Technical Report 72(1998-HPC-072), 電子技術総合研究所, 電子技術総合研究所, aug 1998.

[125] 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. Technical Report 98(1999-DSM-016), nov.

[126] 新平和礼 善明晃由斎藤貴文. ストリーム処理を考慮したレイヤ化されたデータ転送管理システム. In *DEIM Forum 2020 I8-3.*, 2020.

[127] 松田智. アウェアネス情報を取り込んだ仮想空間システムの提案. Unisys 技報:Unisys technology review, Vol. 29, No. 2, p. 102, August 2009.

[128] 松田智, 柴田直樹, 安本慶一, 伊藤実. 仮想空間で実世界のアウェアネス情報を取り込んだ効率よいコミュニケーションを実現するためのフレームワークの提案. Technical Report 16(2007-DPS-130), 奈良先端科学技術大学院大学情報科学研究科, 滋賀大学情報管理学科, 奈良先端科学技術大学院大学情報科学研究科, 奈良先端科学技術大学院大学情報科学研究科, mar 2007.

[129] 凌輔上田, 史暁竹森. 人の動線を予測したポテンシャル法に基づく電動車椅子の歩行者回避アシスト. ロボティクス・メカトロニクス講演会講演概要集, Vol. 2018, No. 0, pp. 2P1–F02, 2018.

[130] 亀岡嵩幸. バーチャル学会の可能性. 応用物理, Vol. 90, No. 2, pp. 126–129, 2021.

[131] 総務省, 東京大学. データ流通プラットフォーム間の連携を実現するための基本的事項. Technical report, 2017.

[132] 松岡大祐, 荒木文明. 大規模シミュレーションデータ可視化における研究の動向. *JAMSTEC Report of Research and Development*, Vol. 13, pp. 35–63, 2011.

[133] 光汰池田, 拓登篠原, 研折本, 慎平松本, 智子健山. ポテンシャルフィールドによる海洋環境考慮をした魚群行動生成 (情報システム研究会 知的情報システム・その他一般). 電気学会研究会資料. IS, Vol. 2020, 34-46・48・51-66, pp. 55–58, oct 2020.

[134] 藤野直明. 第 4 次産業革命とシステムの経済. 横幹, Vol. 14, No. 1, pp. 50–63, 2020.

[135] 富士経済. コネクテッドカー･v2x･自動運転関連市場の将来展望 2020. *IEEE MultiMedia*, 2020.

[136] 平野靖, 鳥脇純一郎. 距離変換を用いた図形の構造解析手法. *Medical Imaging Technology*, Vol. 20, No. 1, p. 13, 2002.

[137] 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.

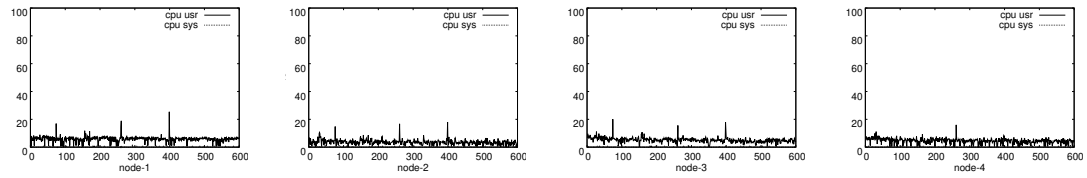
付 録A　付録

A.1　分散システム構成ノードのリソース消費

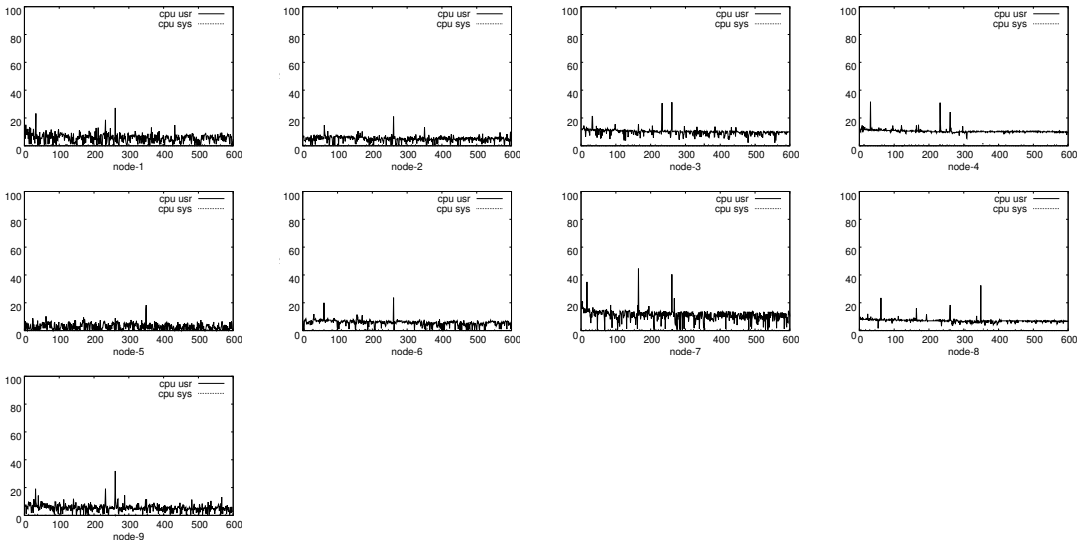
本付録では 3.6.2 節で述べた実験結果データを記す．正方格子状に仮想配置しネットワーク接続したノードネットワークシステムを対象に，ノード数が 2^2〜10^2 台で構成した各システム別に，10 分間の実験中の各ノードのリソース消費量（CPU 利用率，ネットワーク転送レート，メモリ利用量）の変動を計測したものである．いずれの図も横軸の単位は秒である．メモリ消費量は Java のヒープメモリ消費量を Survivor 領域 0 と 1，Eden 領域，Old 領域の使用量を合算して求めたものであり，この値は JVM の GC の影響で定期的に上下することになる．ネットワーク消費量が時間経過とともに漸減していることが確認できるが，これは今回動作させた PCG モデル（3.6.2 節で言及）が，近接相互作用を繰り返してマップデータの内容を更新すると，時間の経過とともにマップデータの内容が一様化する性質があり，そのデータを圧縮して送信する際のサイズが少なくなっていくことが理由である．なお，ノード数が 2^2 台構成の場合は全てのノードが 2 台の近接ノードと相互にネットワーク接続されるが，ノード数が 2^3 台以上の構成の場合は，正方格子上の端に位置するノードは 3 台，角に位置するノードは 2 台，それ以外は 4 台の近接ノードがネットワーク接続されることになり，これに依ってリソースの消費傾向が異なる．したがって，同じノード台数の構成であってもノードに依ってリソース消費傾向が異なる 3 パターンのノードが存在する点に注意されたい．

A.1.1　CPU 利用率（%）

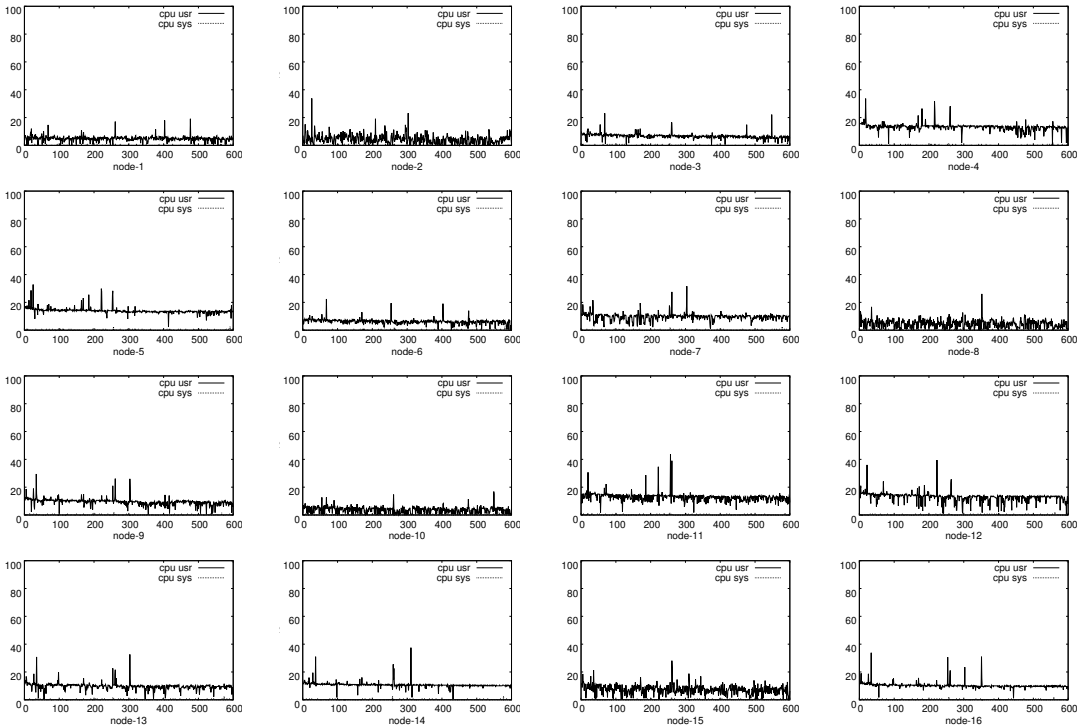
ノード数：2^2



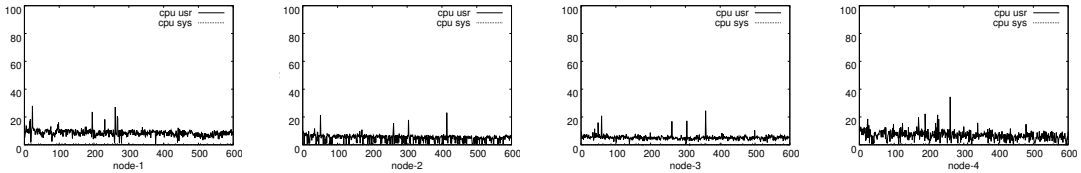
ノード数：3^2

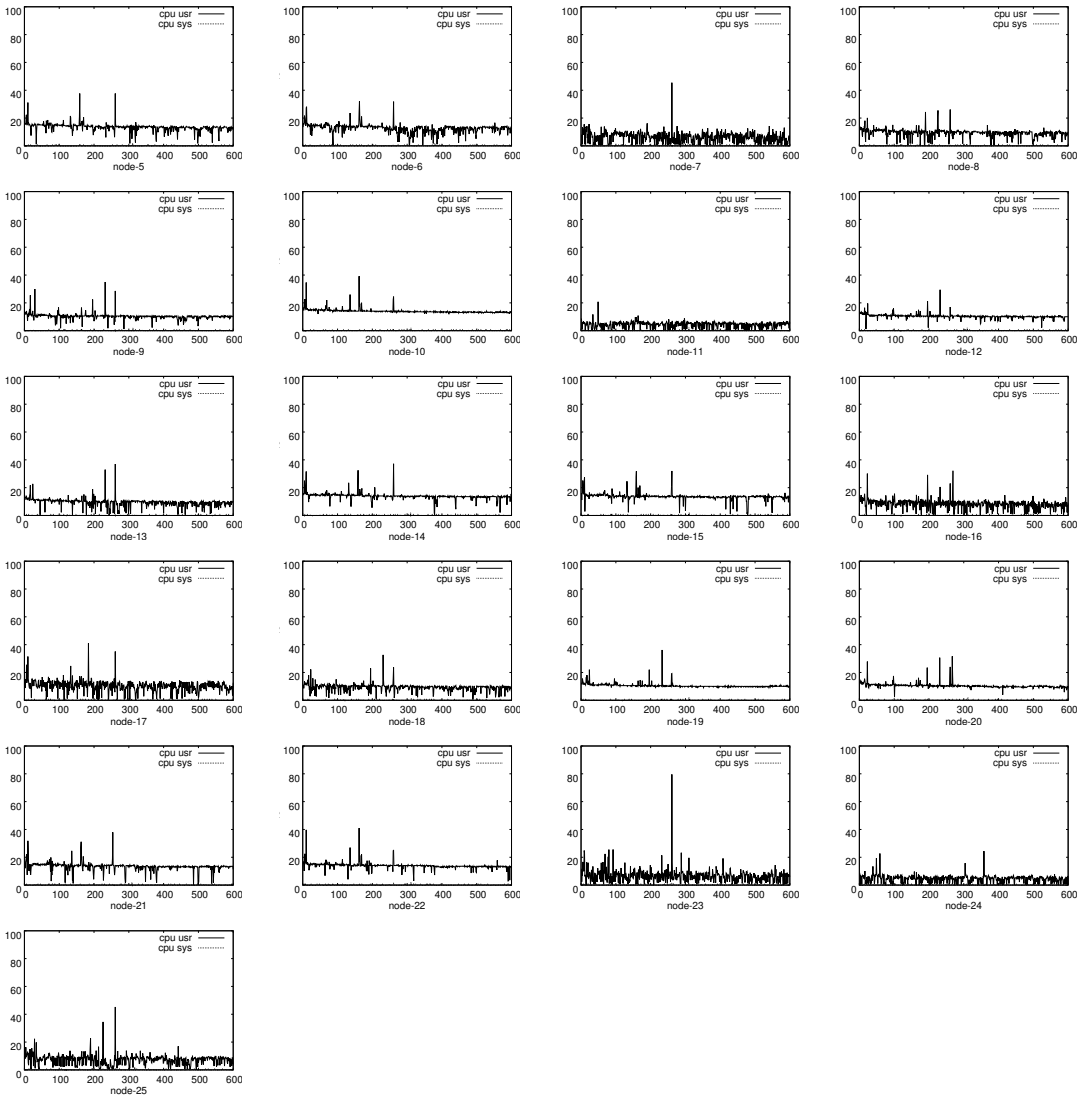


ノード数：4^2

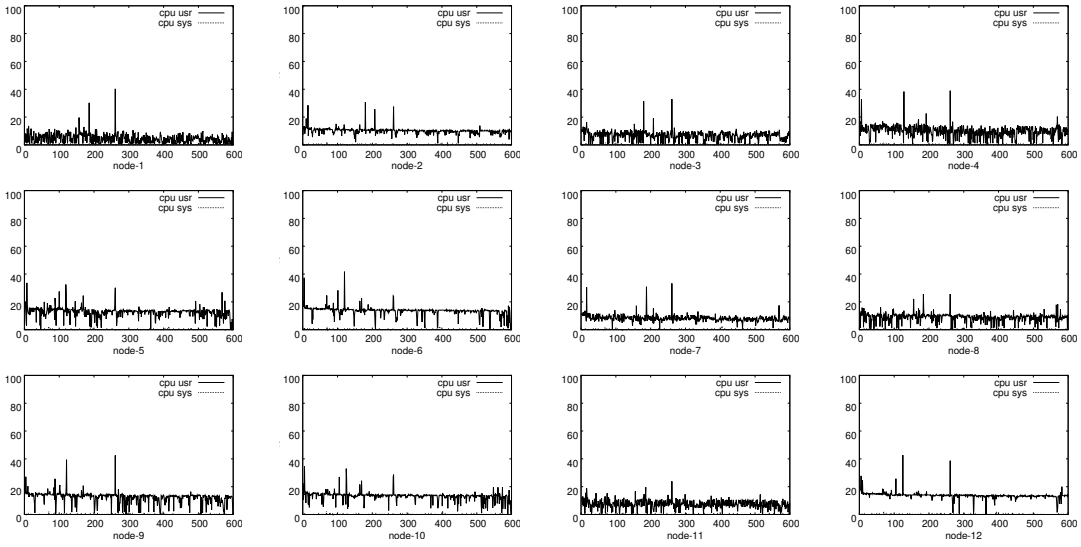


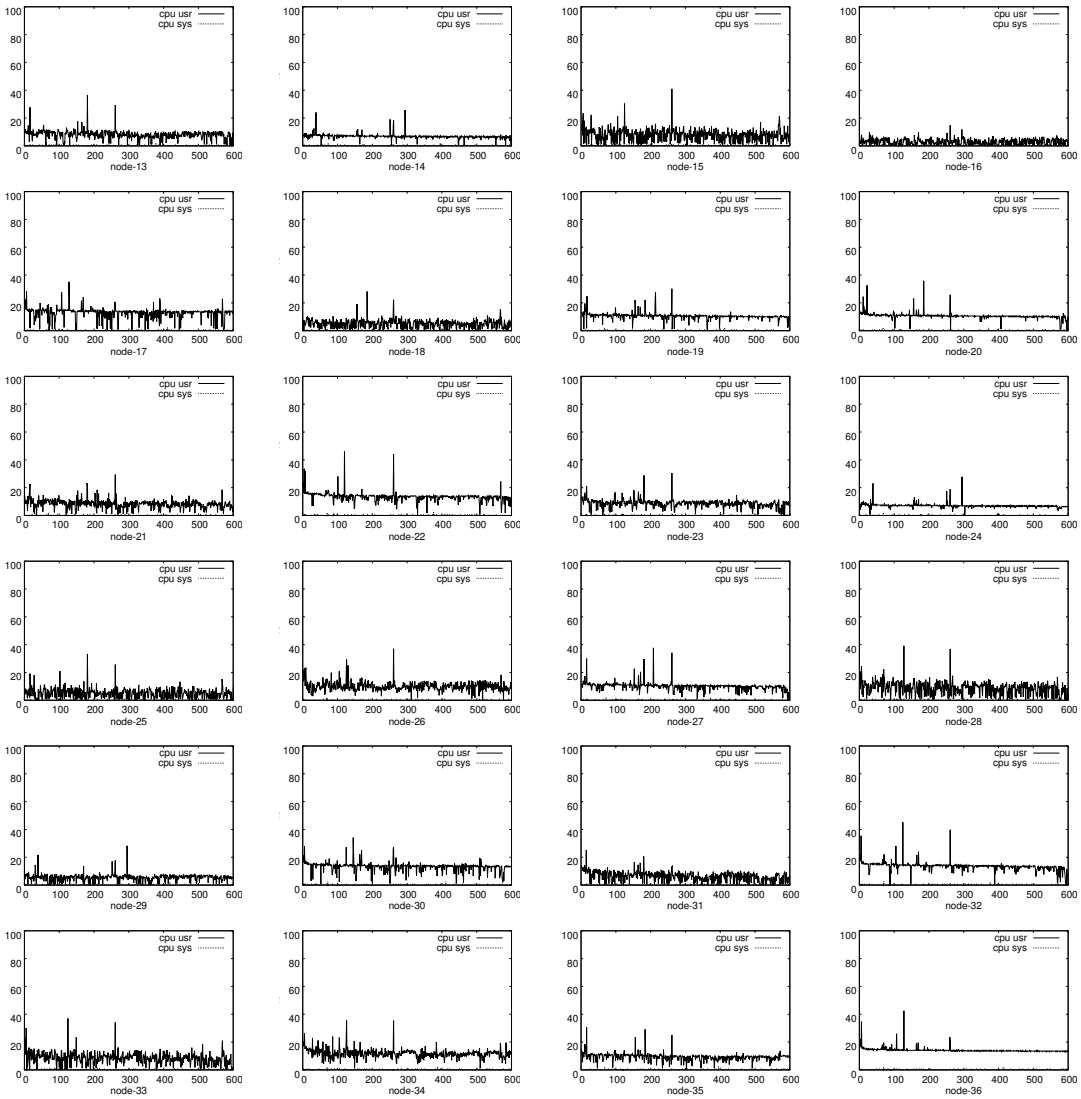
ノード数：5^2



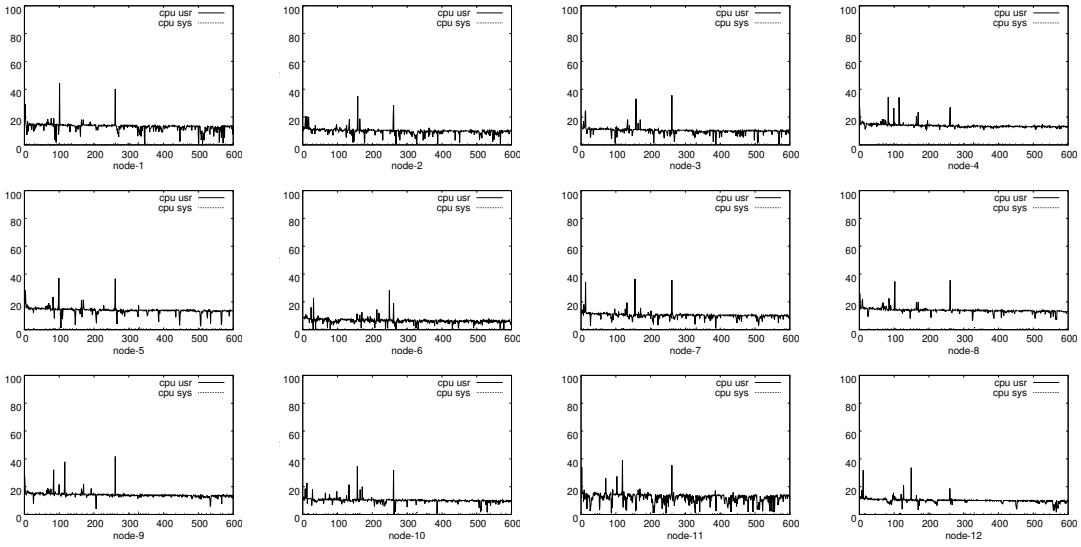

cpu usr
cpu sys
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25

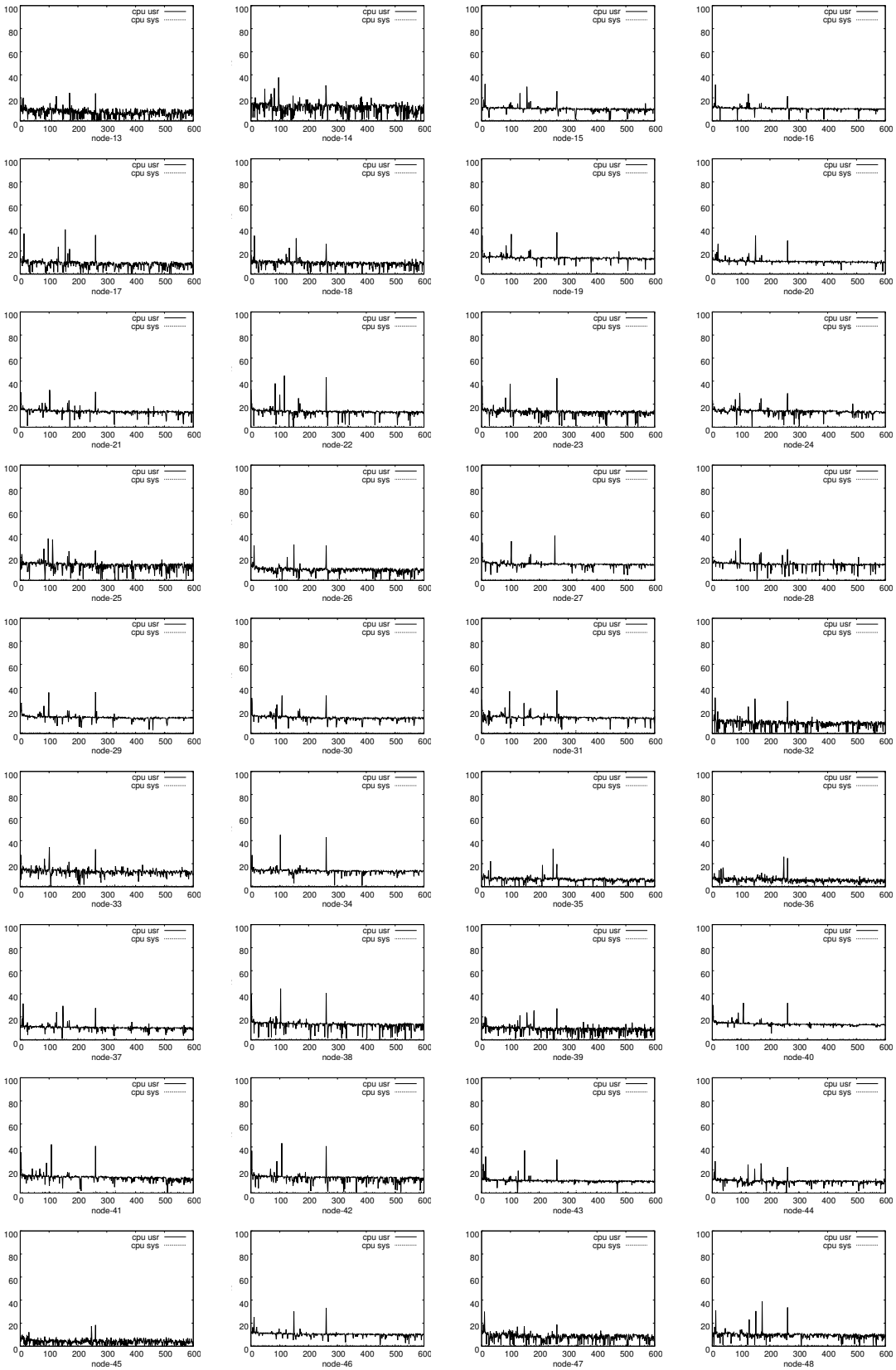
ノード数：6^2

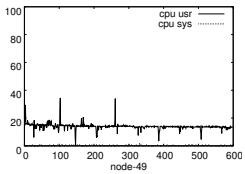

cpu usr
cpu sys
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12


cpu usr
cpu sys
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36

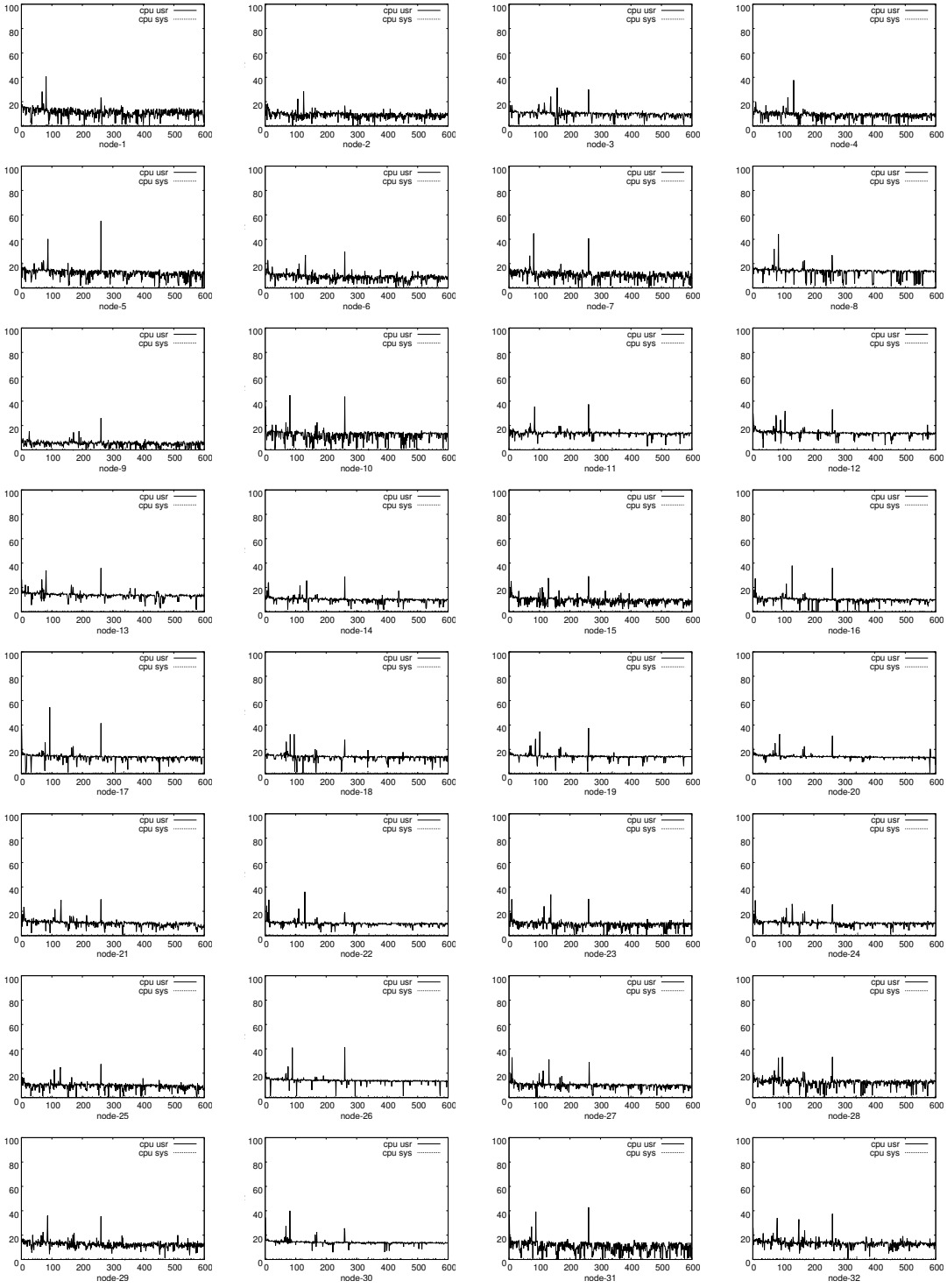
ノード数：7^2

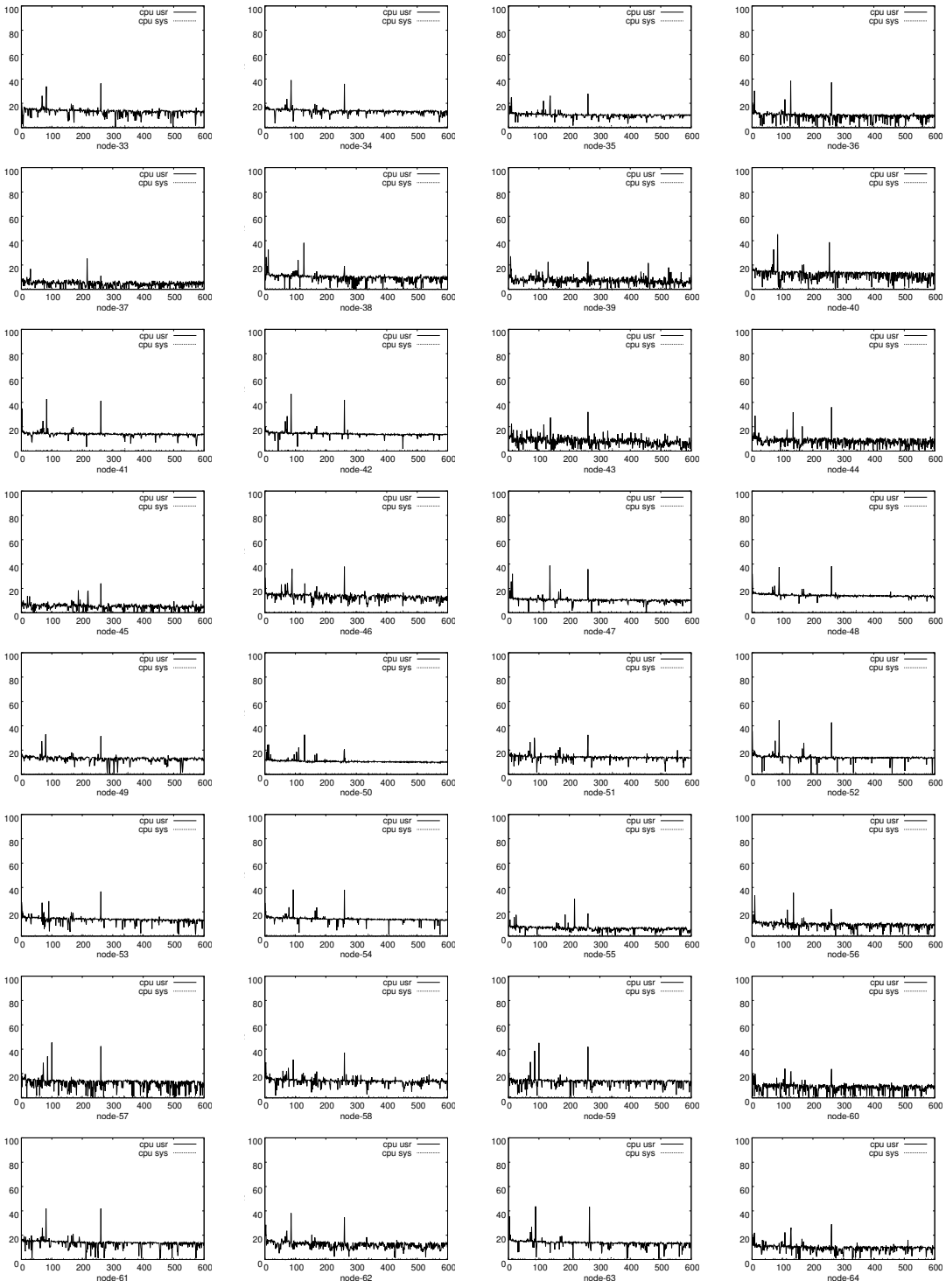

cpu usr
cpu sys
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12


cpu usr
cpu sys
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48

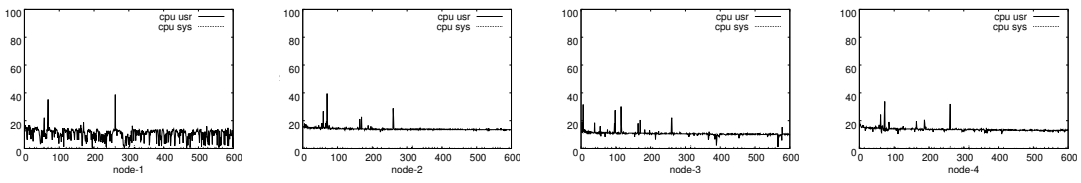

cpu usr
cpu sys
node-49

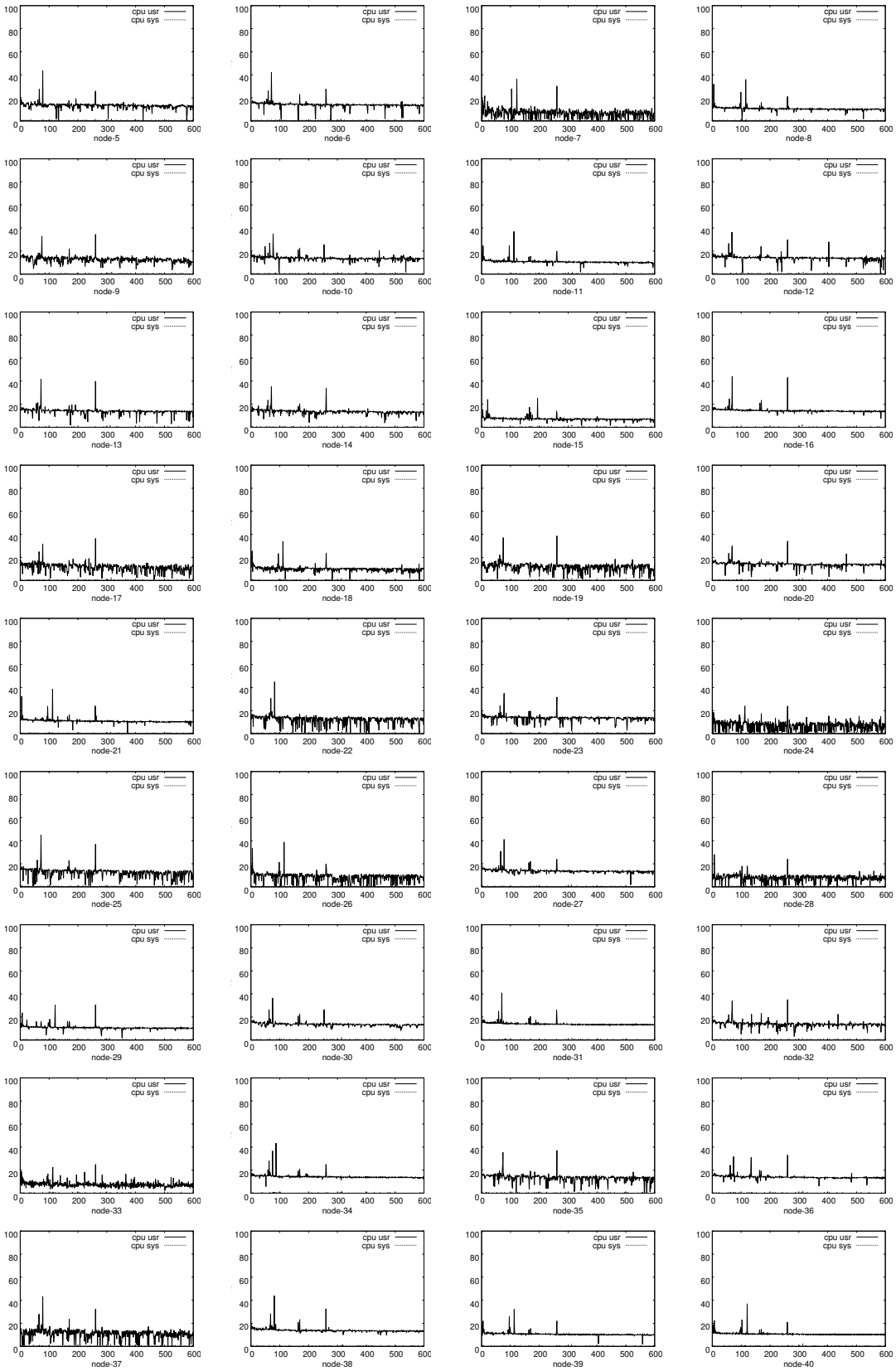
ノード数：8^2

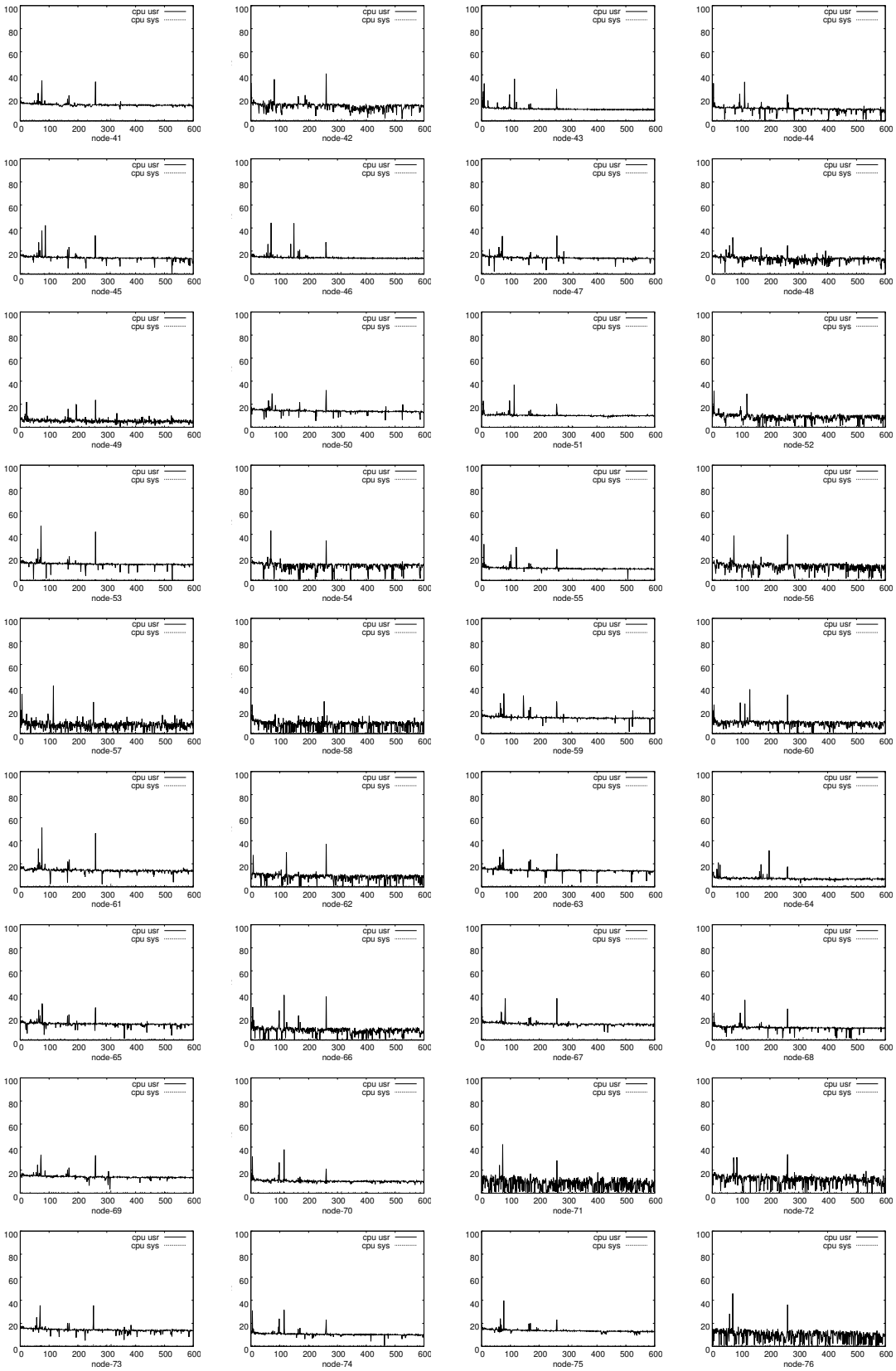

cpu usr
cpu sys
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32

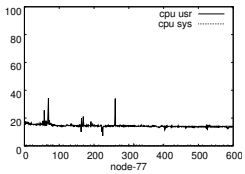
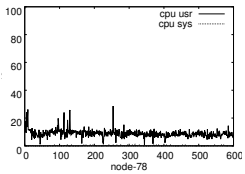
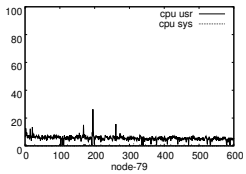
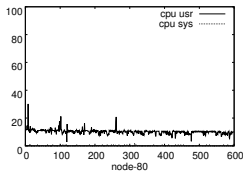
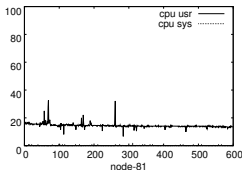

cpu usr
cpu sys
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64

ノード数：9^2

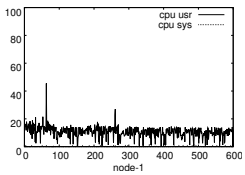
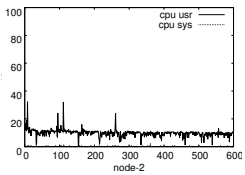
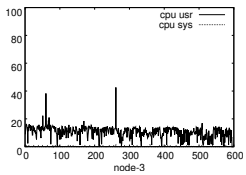
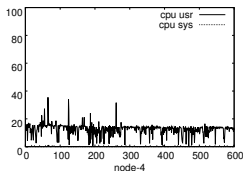
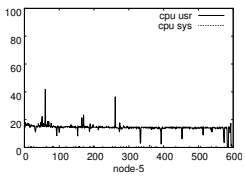
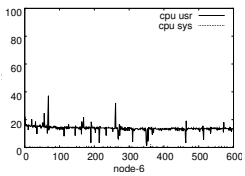
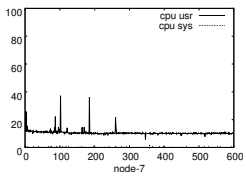
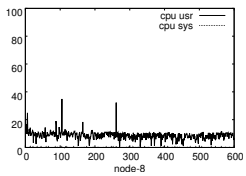
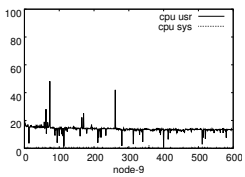
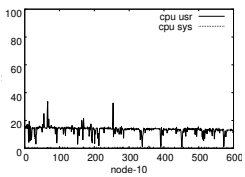
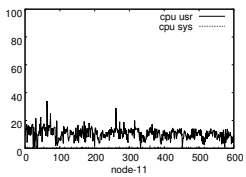
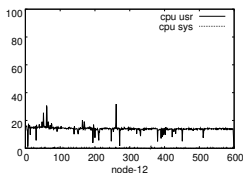
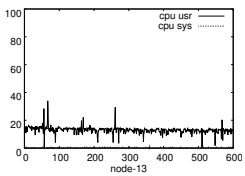
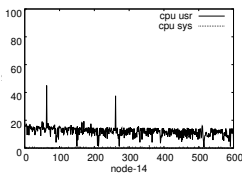
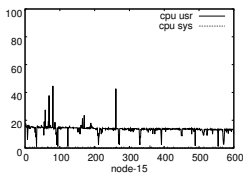
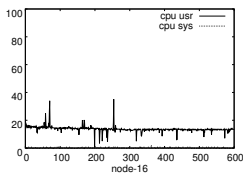
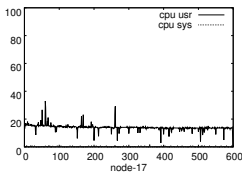
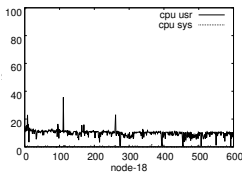
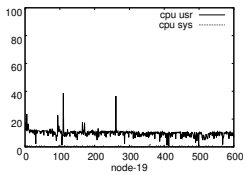
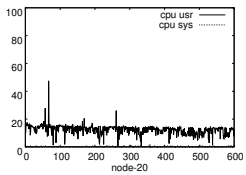
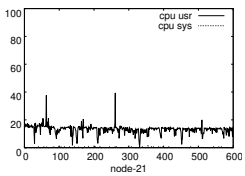
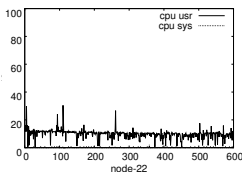
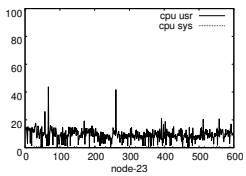
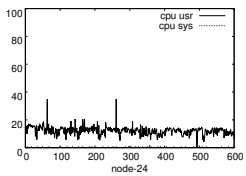
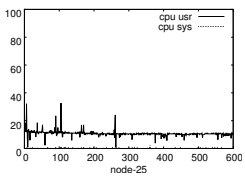
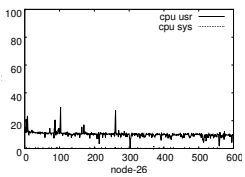
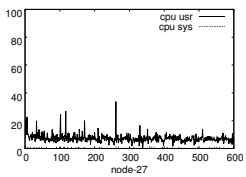
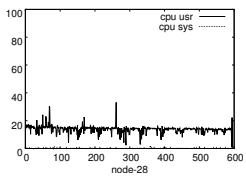

cpu usr
cpu sys
node-1
node-2
node-3
node-4

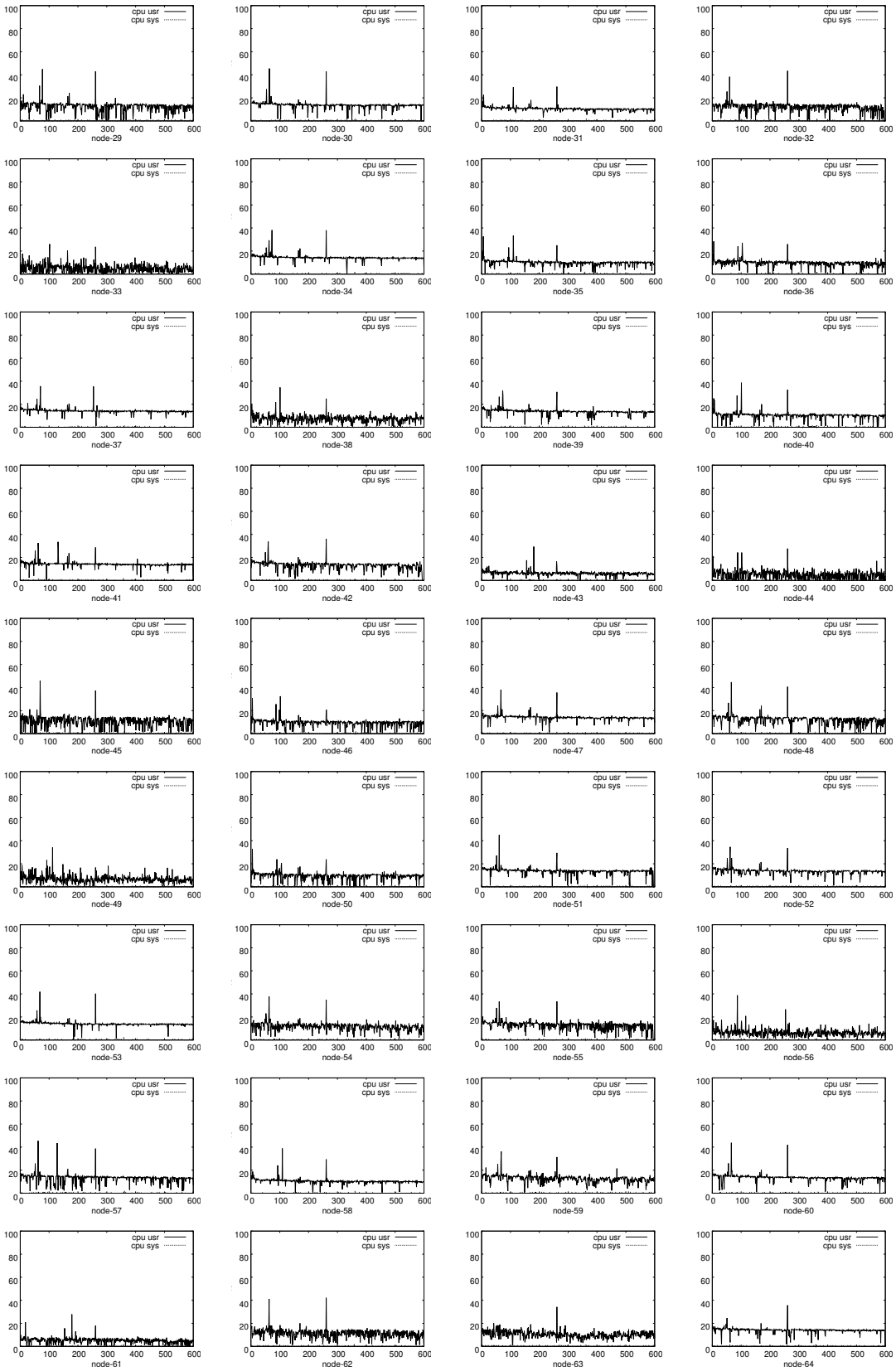

cpu usr
cpu sys
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40

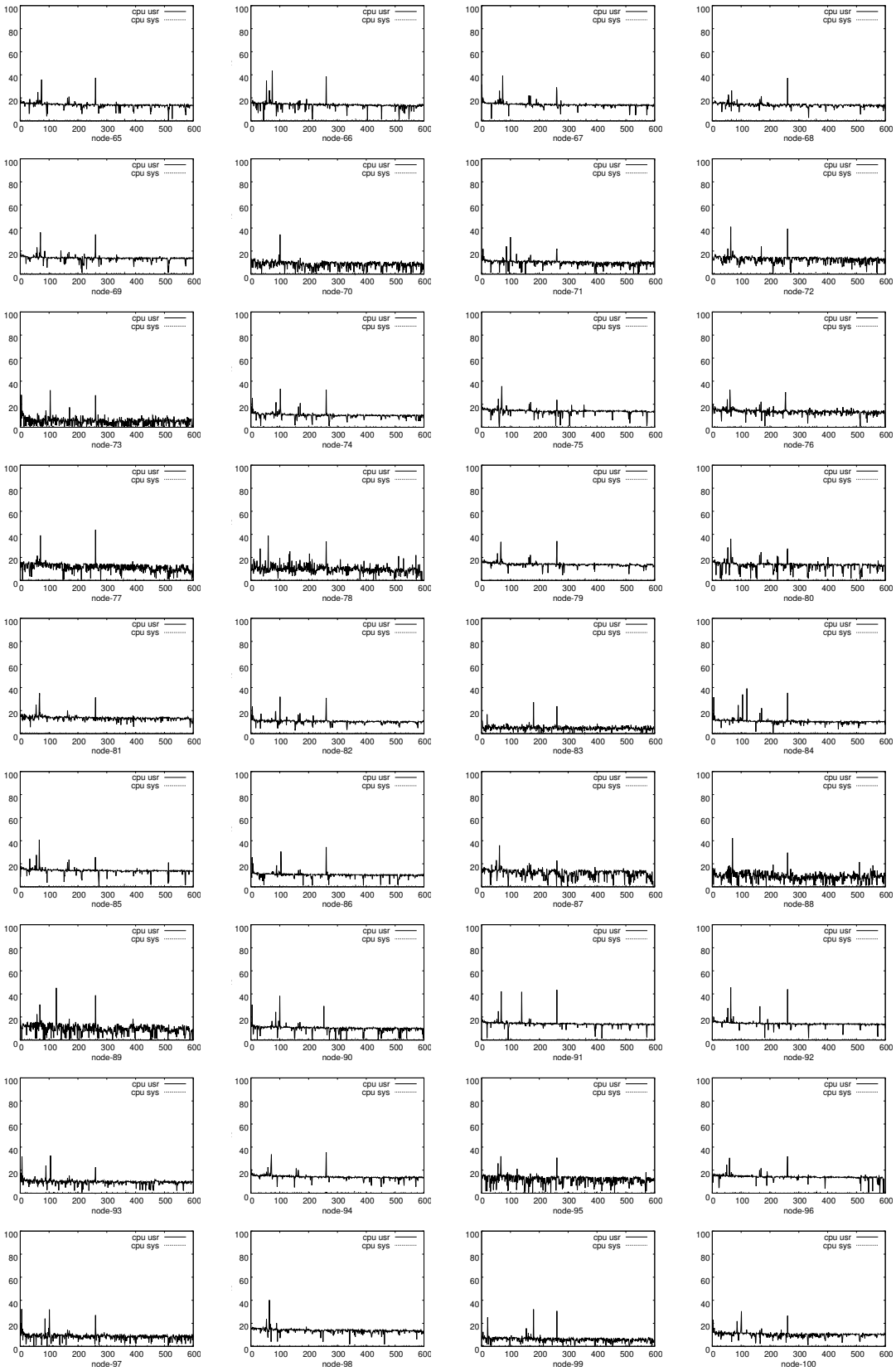

cpu usr
cpu sys
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64
node-65
node-66
node-67
node-68
node-69
node-70
node-71
node-72
node-73
node-74
node-75
node-76


cpu usr
cpu sys
node-77

cpu usr
cpu sys
node-78

cpu usr
cpu sys
node-79

cpu usr
cpu sys
node-80

cpu usr
cpu sys
node-81

ノード数：10^2

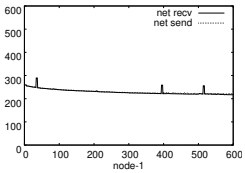

cpu usr
cpu sys
node-1

cpu usr
cpu sys
node-2

cpu usr
cpu sys
node-3

cpu usr
cpu sys
node-4

cpu usr
cpu sys
node-5

cpu usr
cpu sys
node-6

cpu usr
cpu sys
node-7

cpu usr
cpu sys
node-8

cpu usr
cpu sys
node-9

cpu usr
cpu sys
node-10

cpu usr
cpu sys
node-11

cpu usr
cpu sys
node-12

cpu usr
cpu sys
node-13

cpu usr
cpu sys
node-14

cpu usr
cpu sys
node-15

cpu usr
cpu sys
node-16

cpu usr
cpu sys
node-17

cpu usr
cpu sys
node-18

cpu usr
cpu sys
node-19

cpu usr
cpu sys
node-20

cpu usr
cpu sys
node-21

cpu usr
cpu sys
node-22

cpu usr
cpu sys
node-23

cpu usr
cpu sys
node-24

cpu usr
cpu sys
node-25

cpu usr
cpu sys
node-26

cpu usr
cpu sys
node-27

cpu usr
cpu sys
node-28

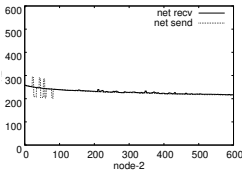

cpu usr
cpu sys
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64

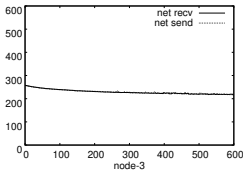

cpu usr
cpu sys
node-65
node-66
node-67
node-68
node-69
node-70
node-71
node-72
node-73
node-74
node-75
node-76
node-77
node-78
node-79
node-80
node-81
node-82
node-83
node-84
node-85
node-86
node-87
node-88
node-89
node-90
node-91
node-92
node-93
node-94
node-95
node-96
node-97
node-98
node-99
node-100

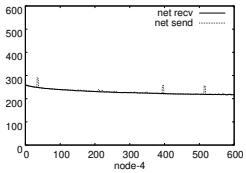
A.1.2　ネットワーク転送レート（KB/s）

ノード数：2^2

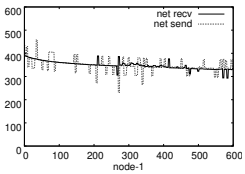


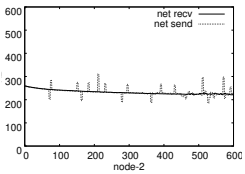


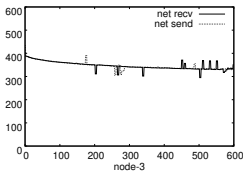


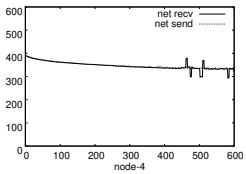


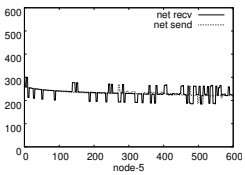
ノード数：3^2

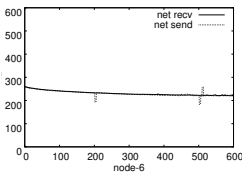


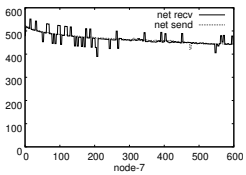


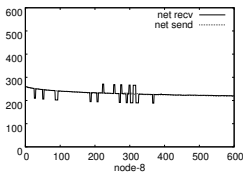


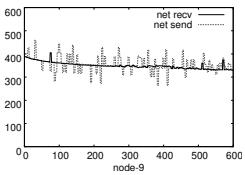




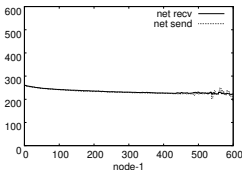


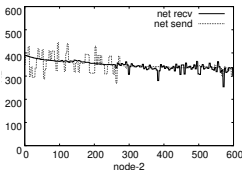


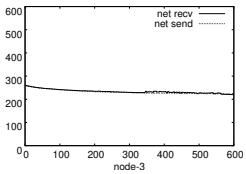


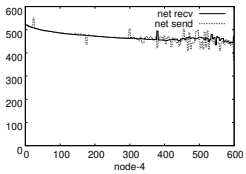


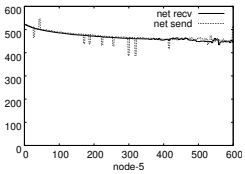
ノード数：4^2

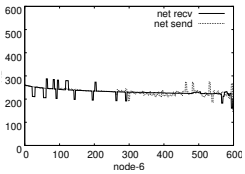


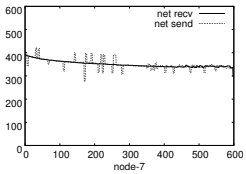


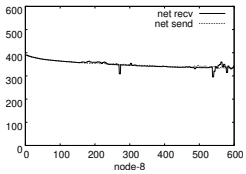


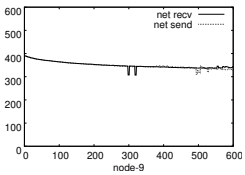


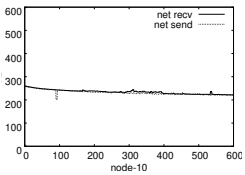


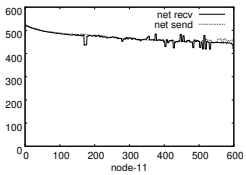


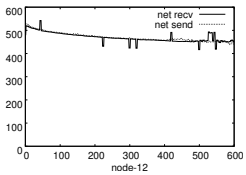


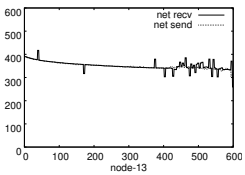


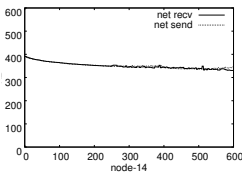


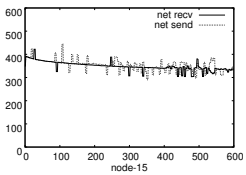


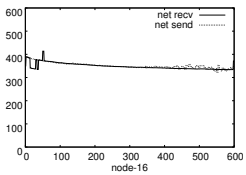




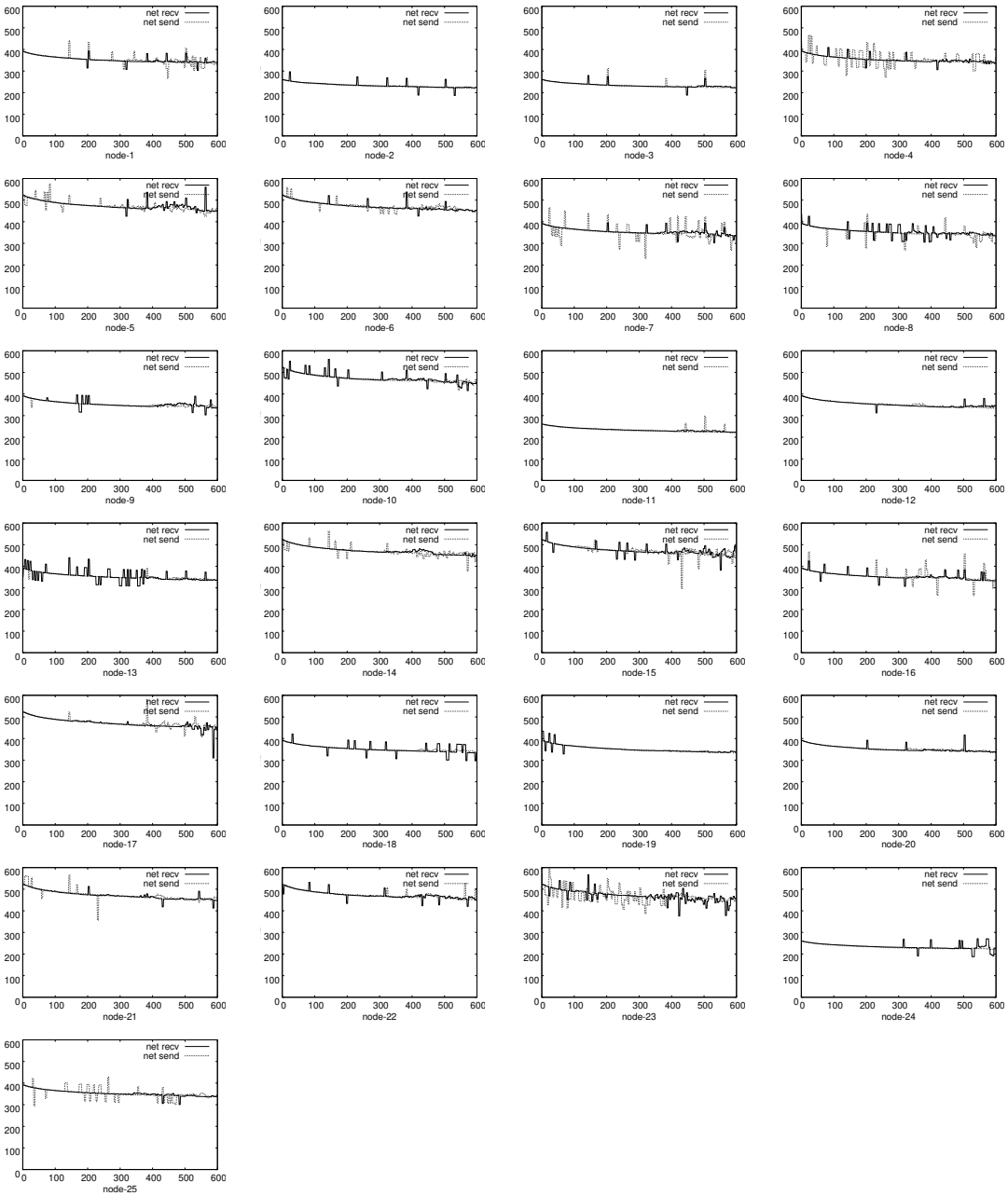




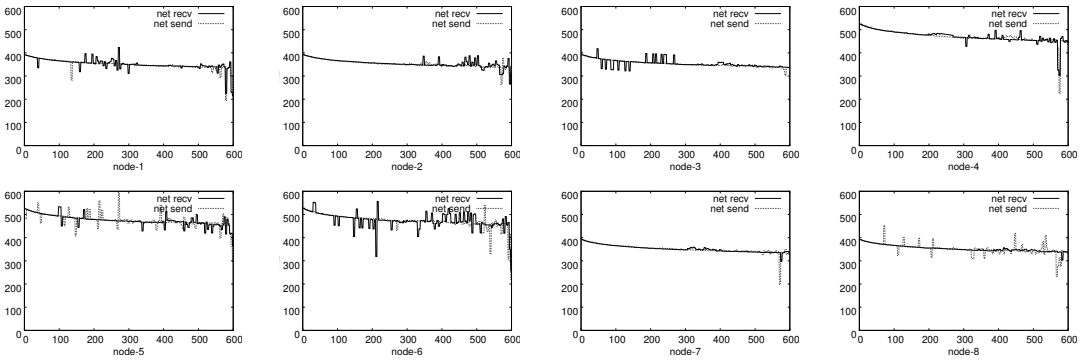


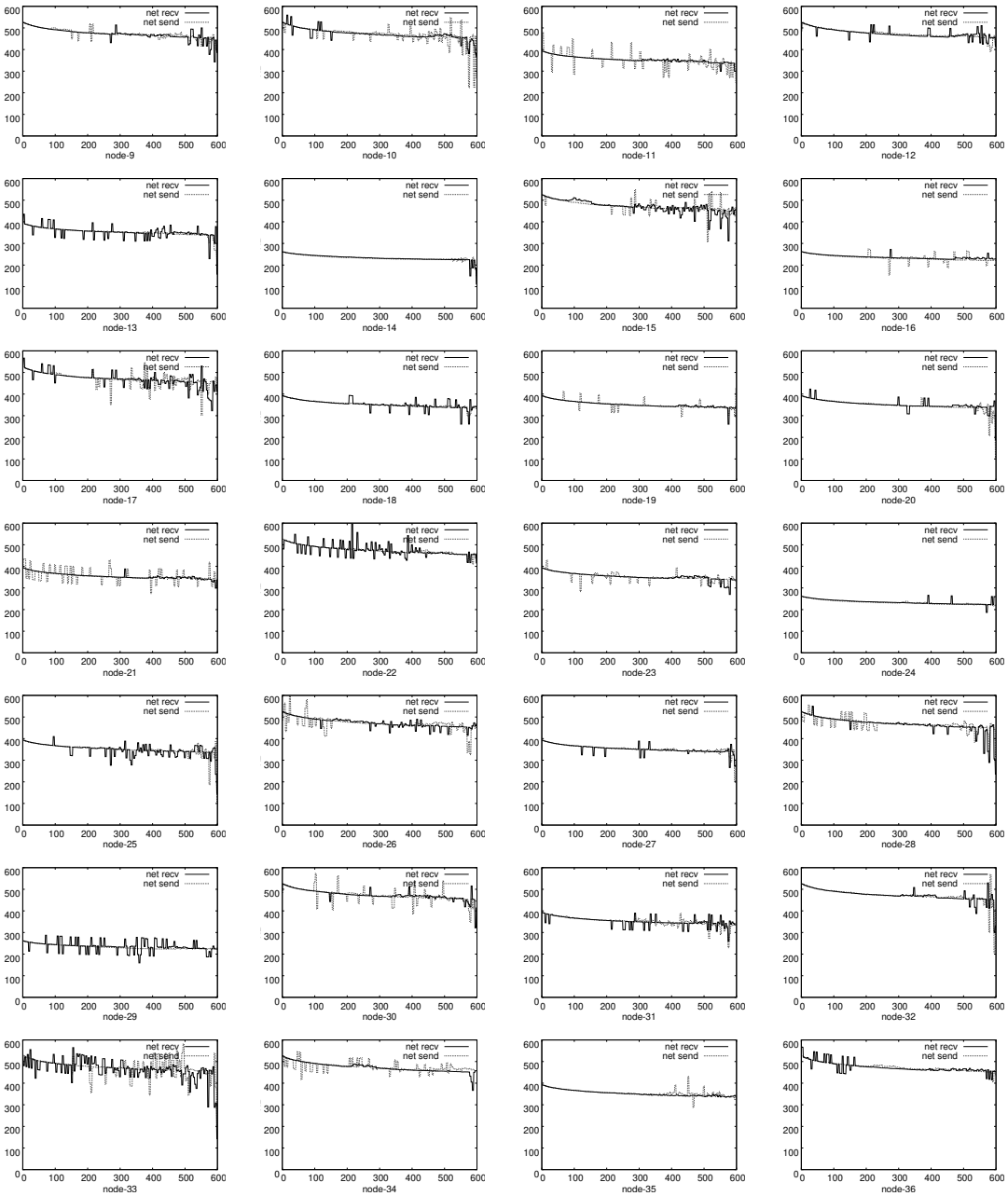


ノード数：5^2

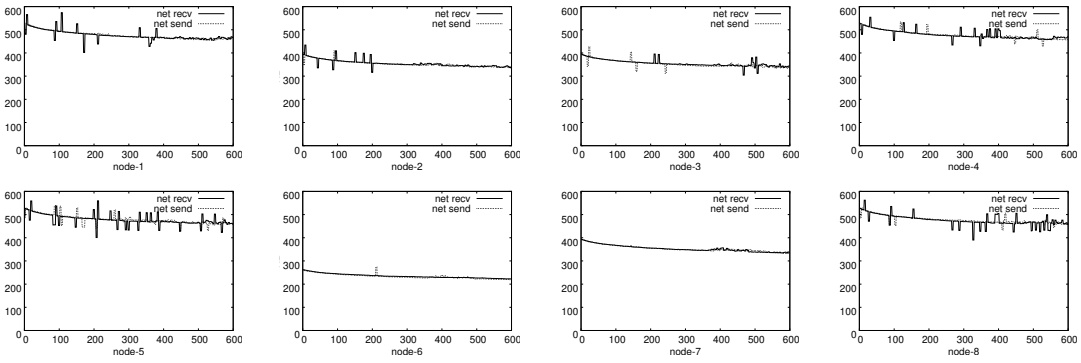


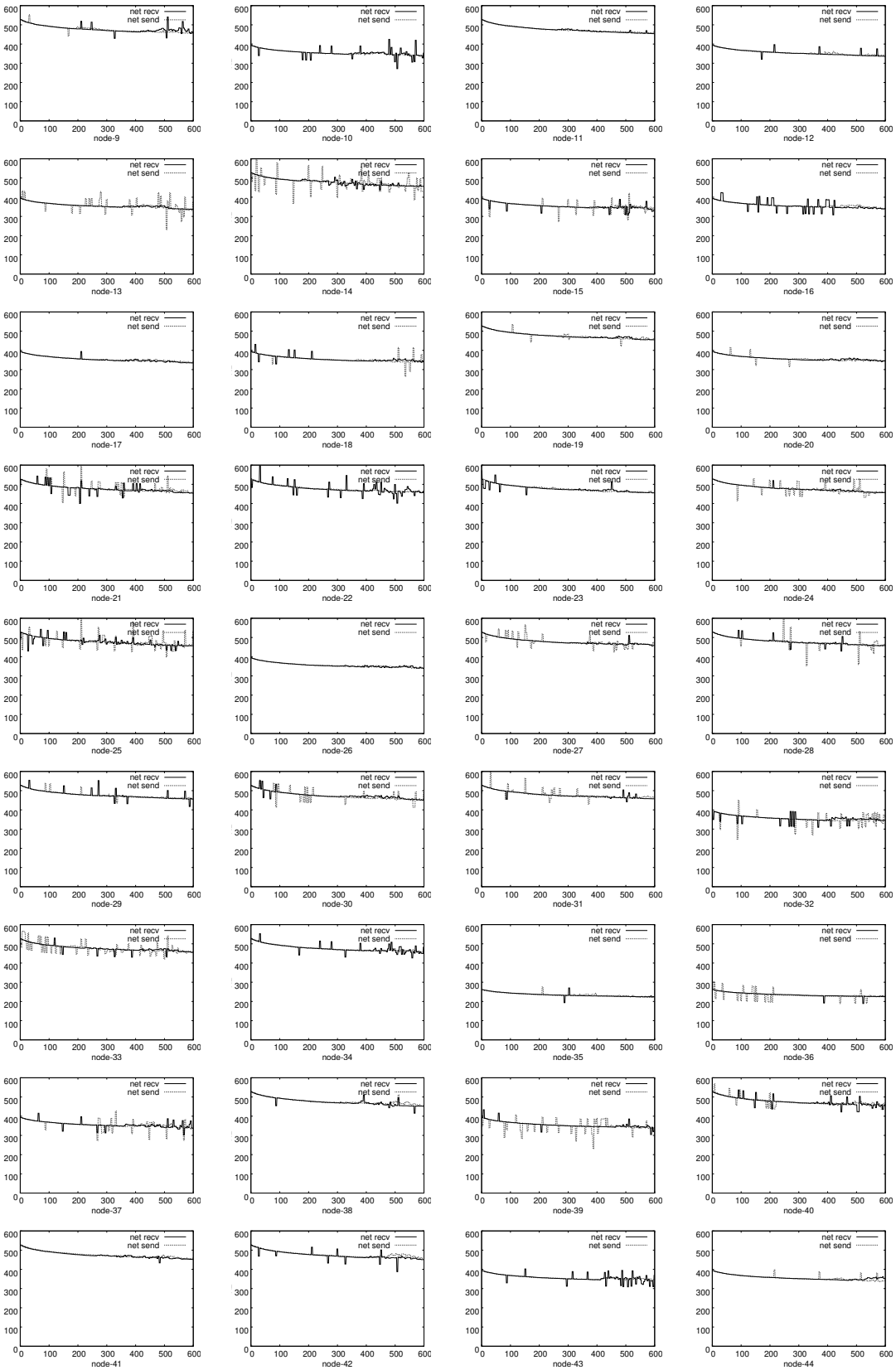
ノード数：6^2

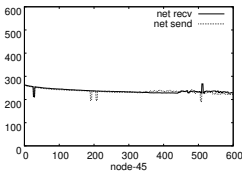
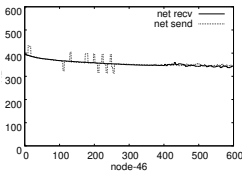
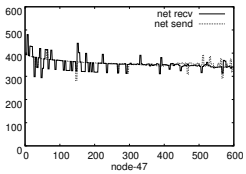
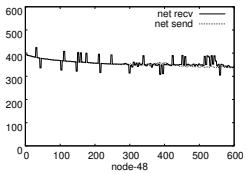
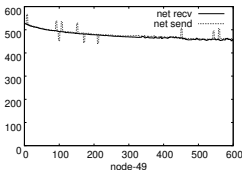



net recv
net send
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36

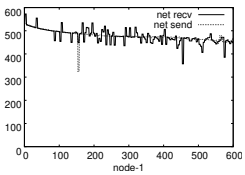
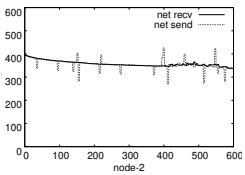
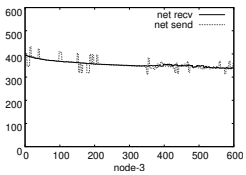
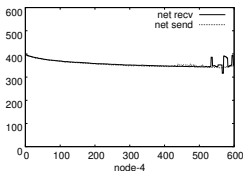
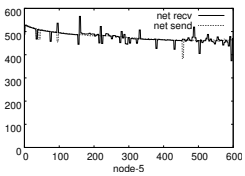
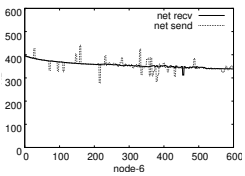
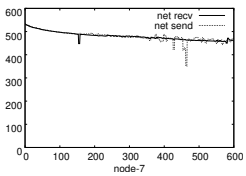
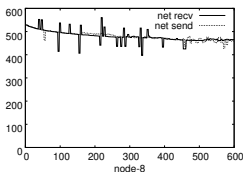
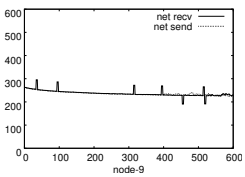
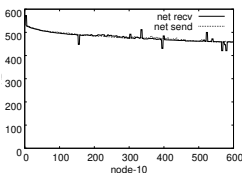
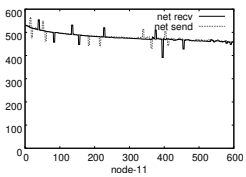
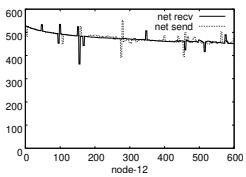
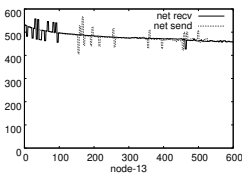
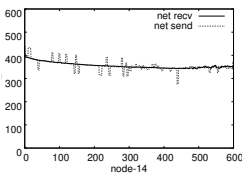
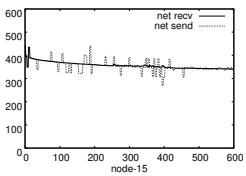
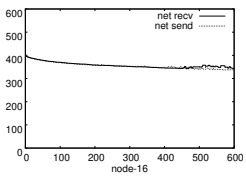
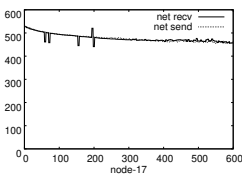
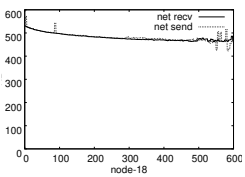
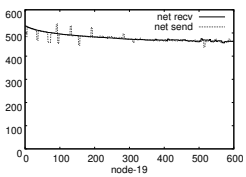
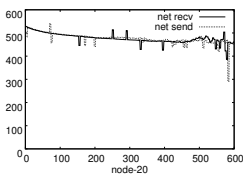
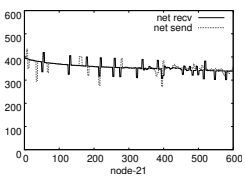
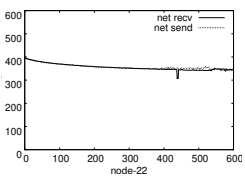
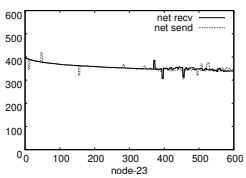
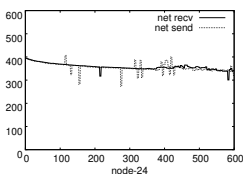
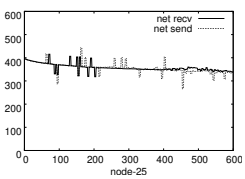
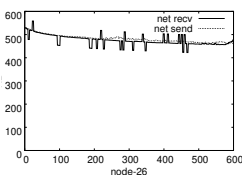
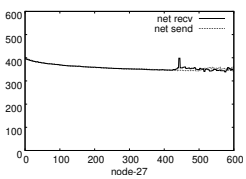
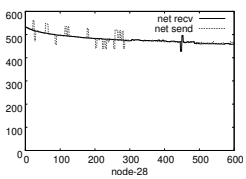
ノード数：7^2

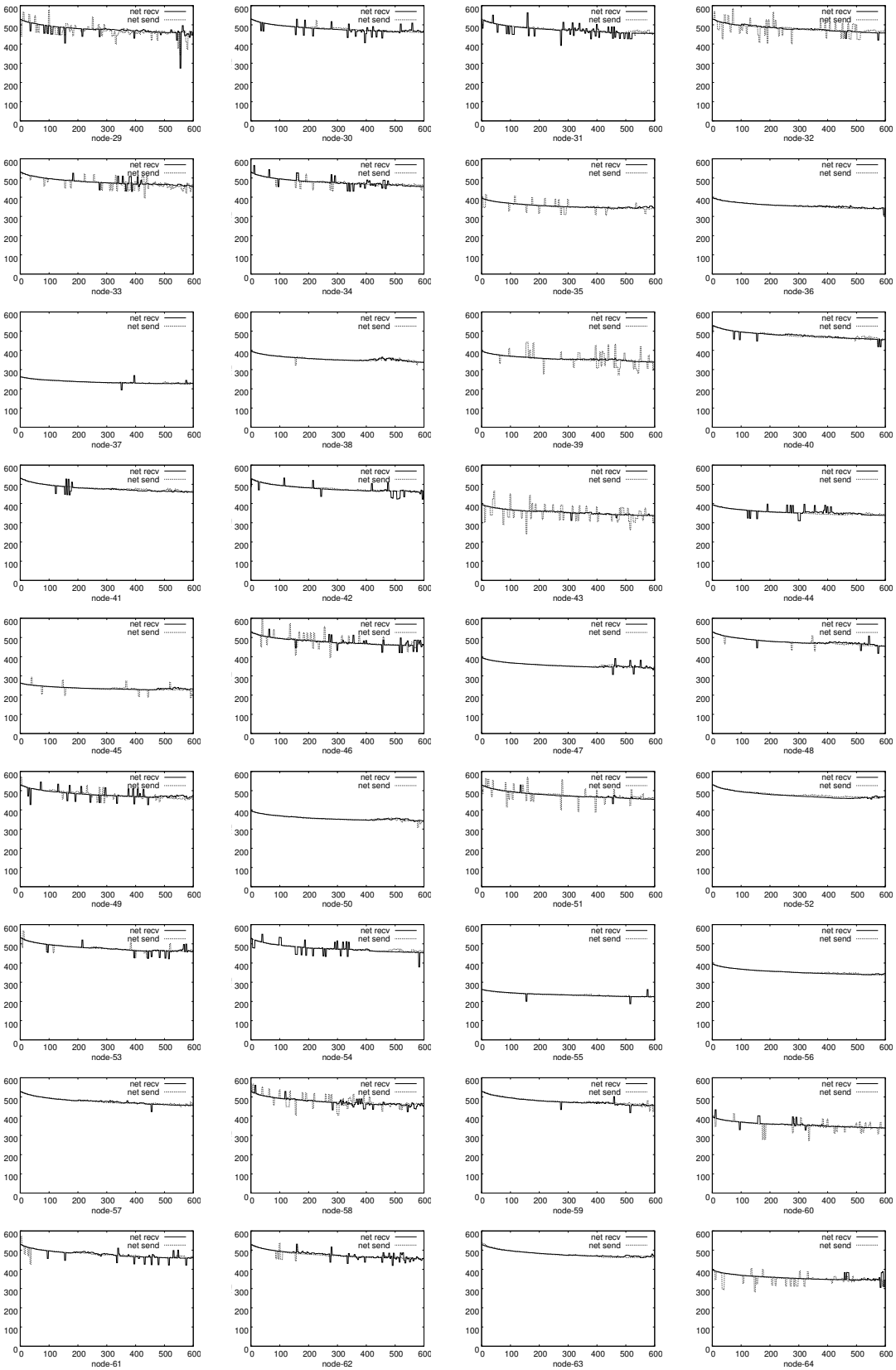

net recv
net send
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8


net recv
net send
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44

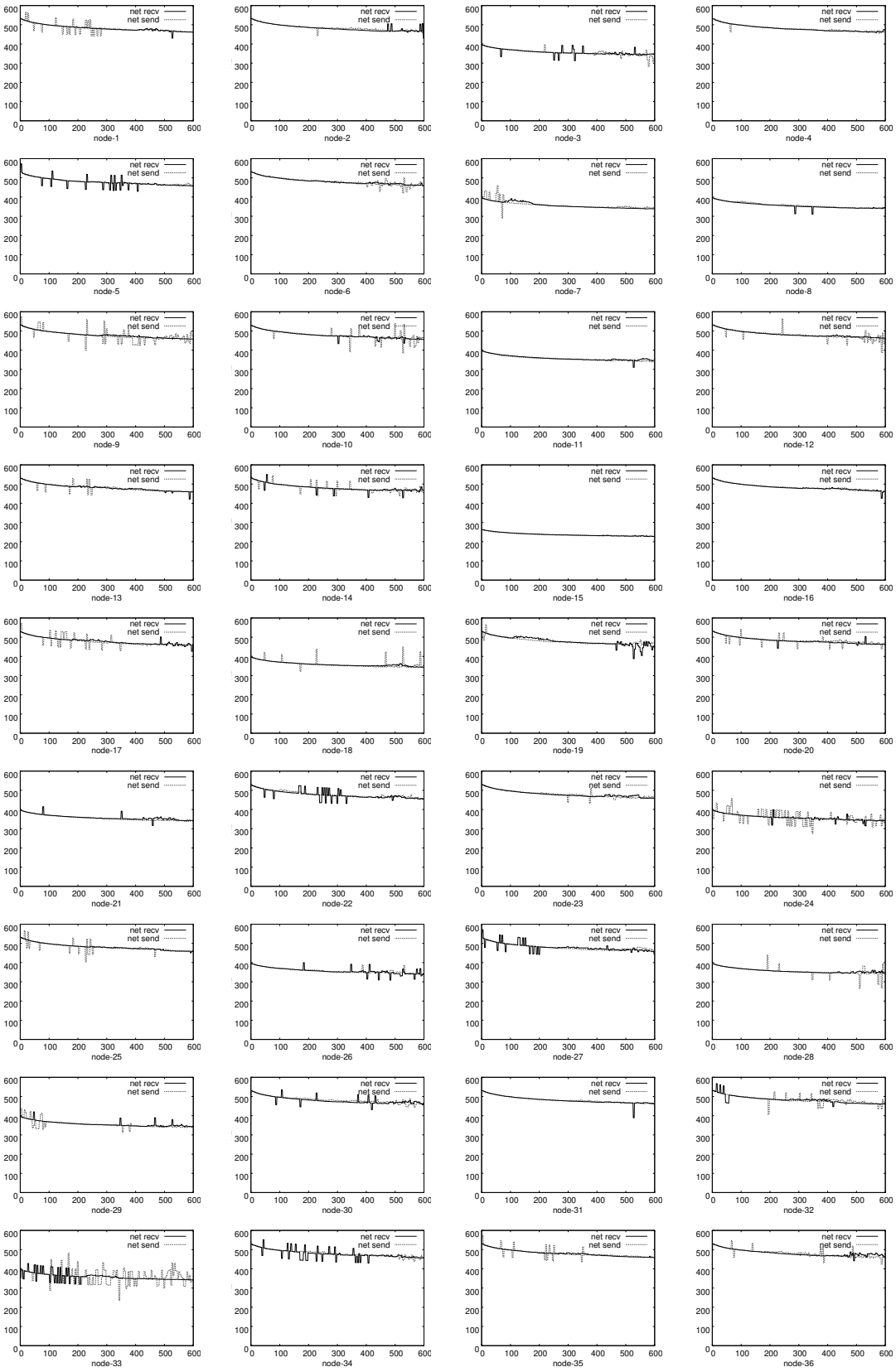

net recv
net send
node-45

net recv
net send
node-46

net recv
net send
node-47

net recv
net send
node-48

net recv
net send
node-49

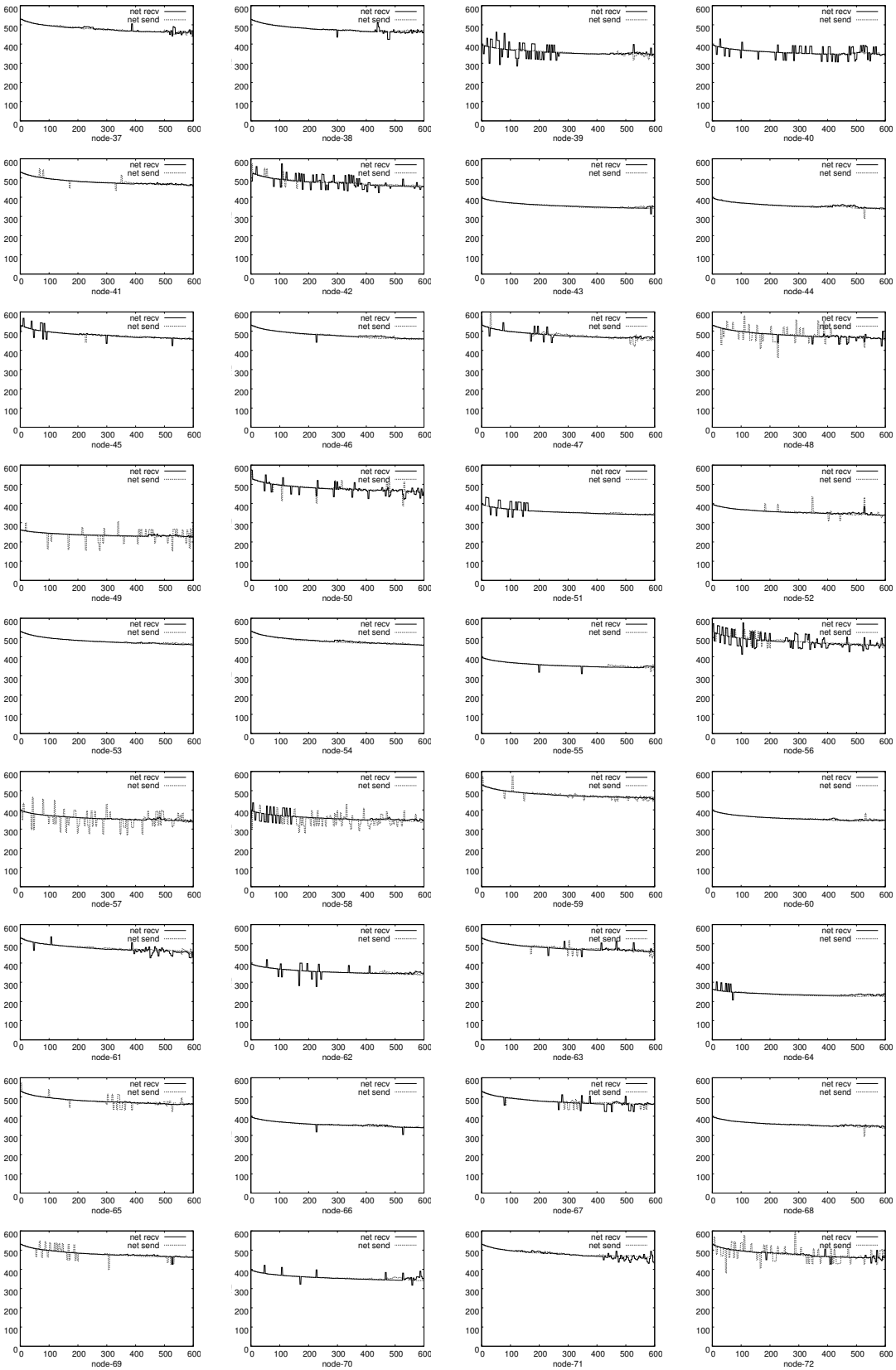
ノード数：8^2

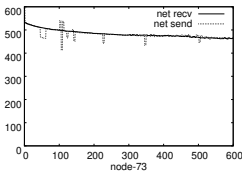
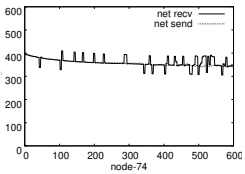
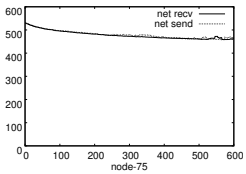
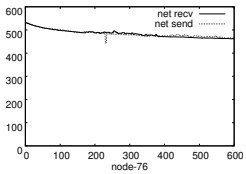
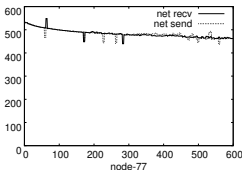
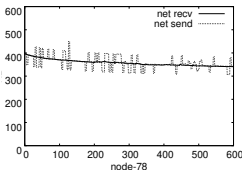
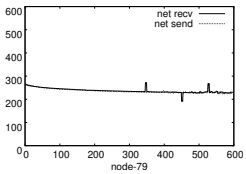
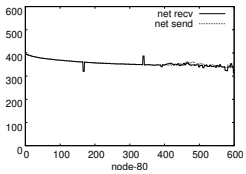
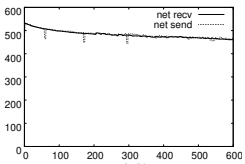

net recv
net send
node-1

net recv
net send
node-2

net recv
net send
node-3

net recv
net send
node-4

net recv
net send
node-5

net recv
net send
node-6

net recv
net send
node-7

net recv
net send
node-8

net recv
net send
node-9

net recv
net send
node-10

net recv
net send
node-11

net recv
net send
node-12

net recv
net send
node-13

net recv
net send
node-14

net recv
net send
node-15

net recv
net send
node-16

net recv
net send
node-17

net recv
net send
node-18

net recv
net send
node-19

net recv
net send
node-20

net recv
net send
node-21

net recv
net send
node-22

net recv
net send
node-23

net recv
net send
node-24

net recv
net send
node-25

net recv
net send
node-26

net recv
net send
node-27

net recv
net send
node-28


net recv
net send
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64

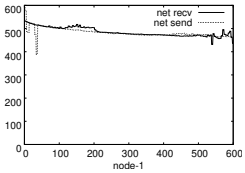
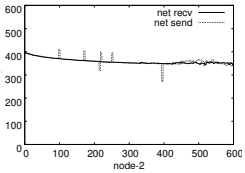
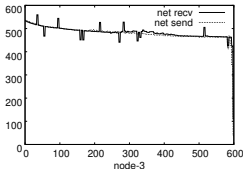
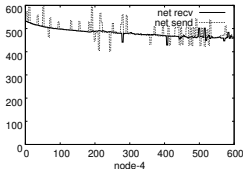
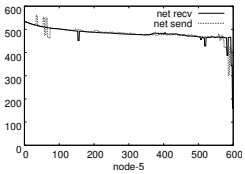
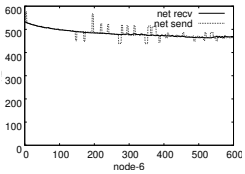
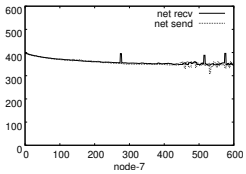
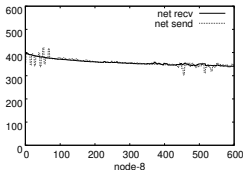
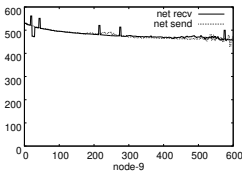
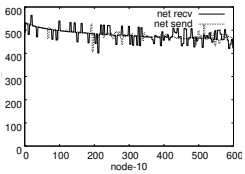
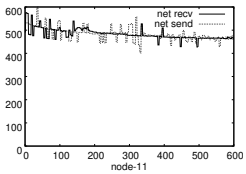
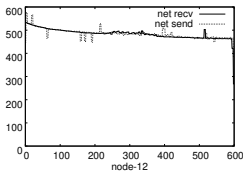
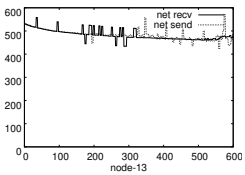
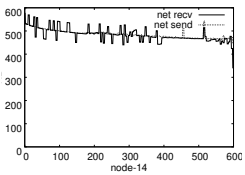
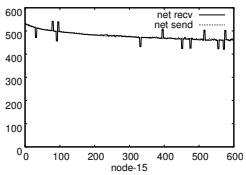
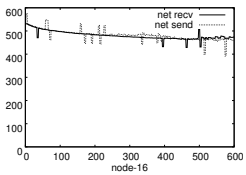
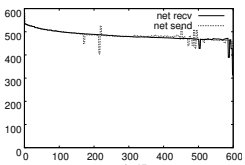
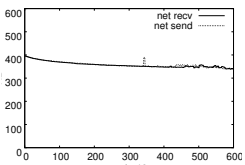
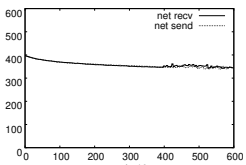
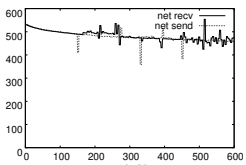
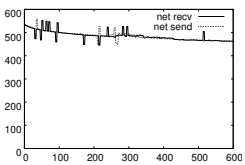
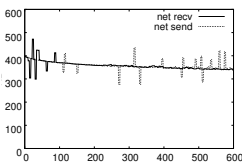
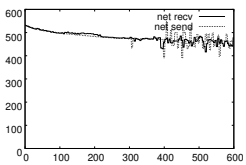
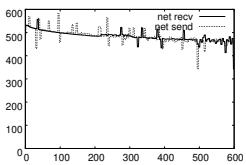
ノード数：9^2

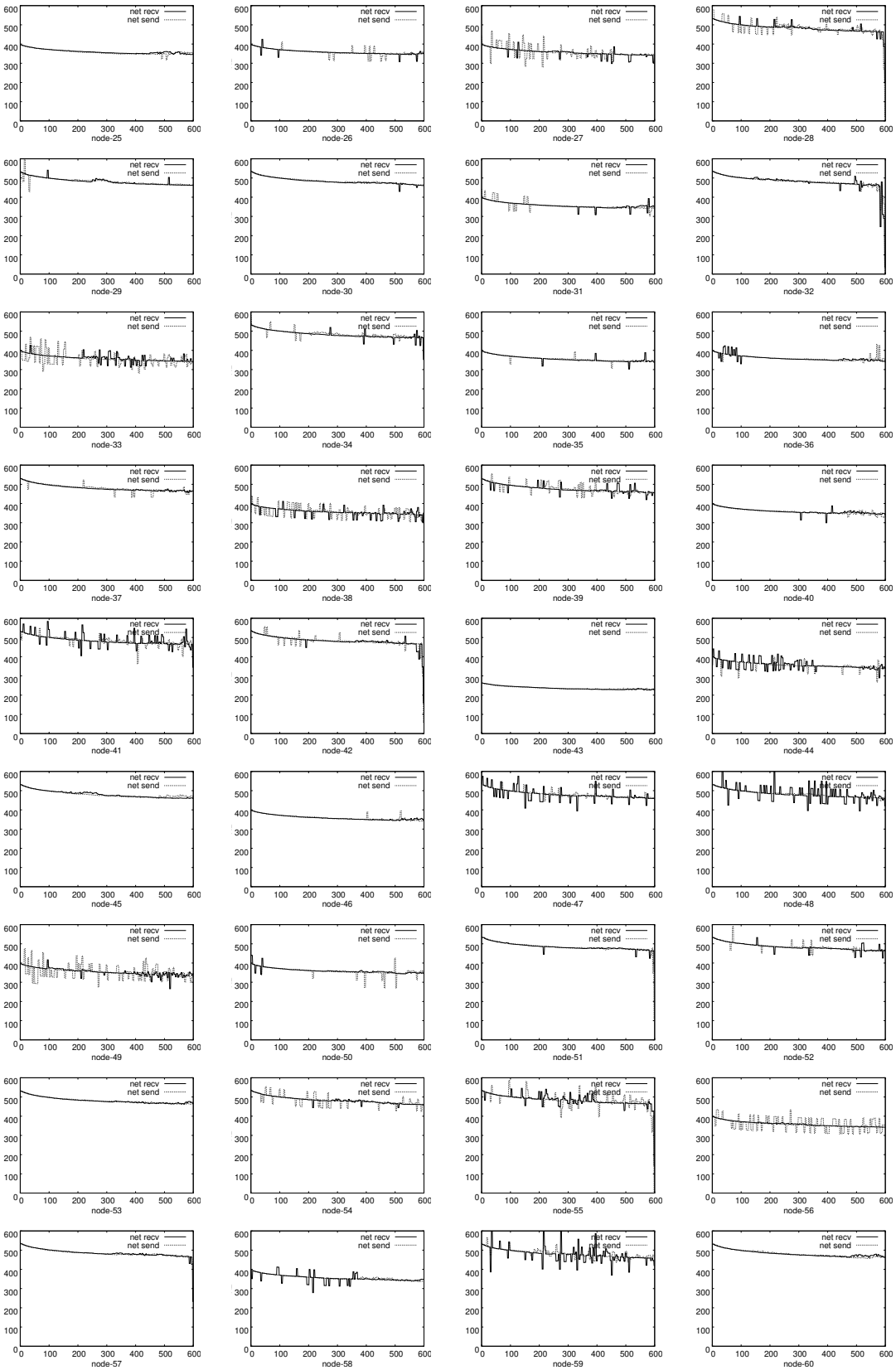

net recv
net send
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36

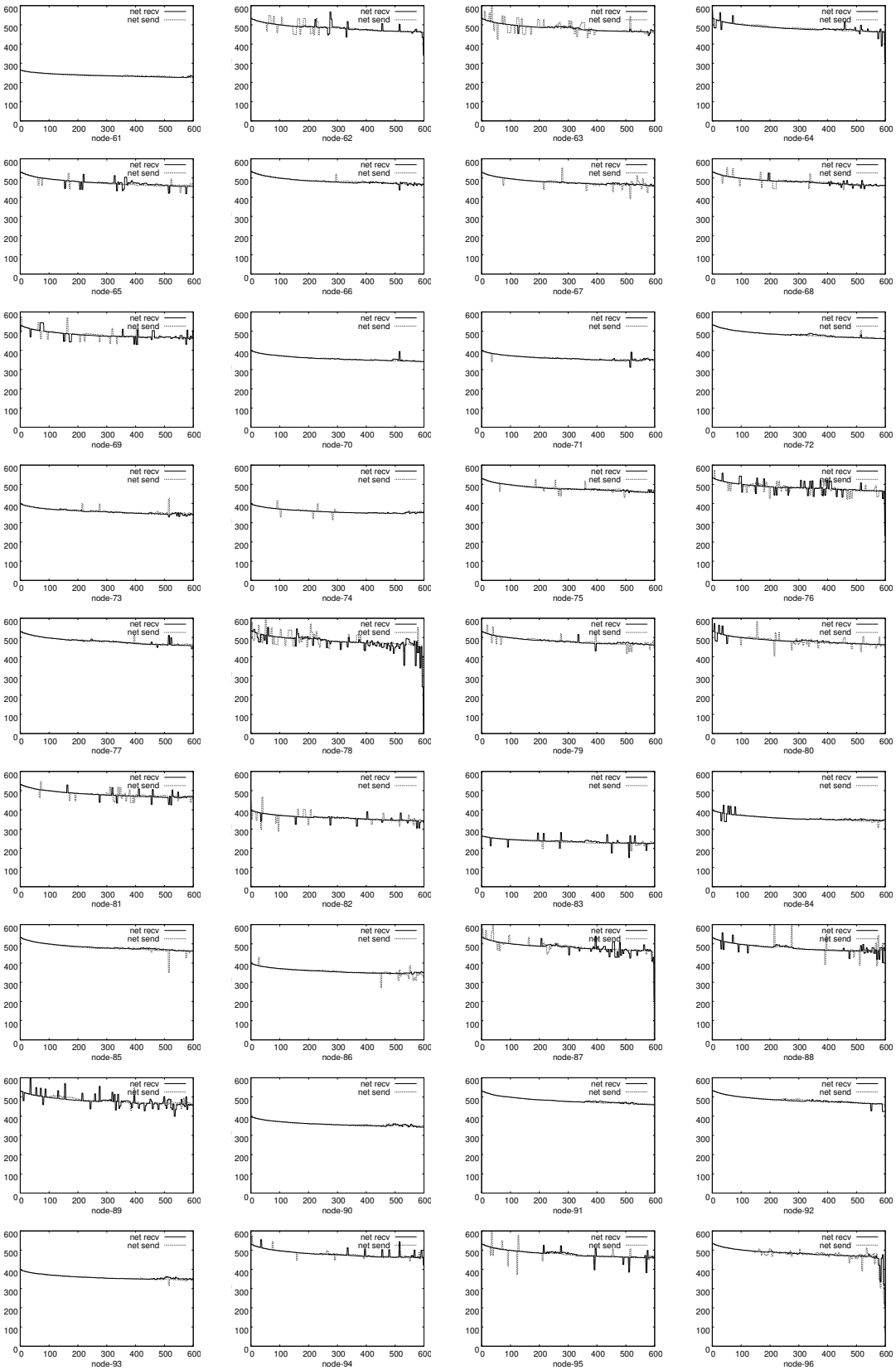

net recv
net send
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64
node-65
node-66
node-67
node-68
node-69
node-70
node-71
node-72

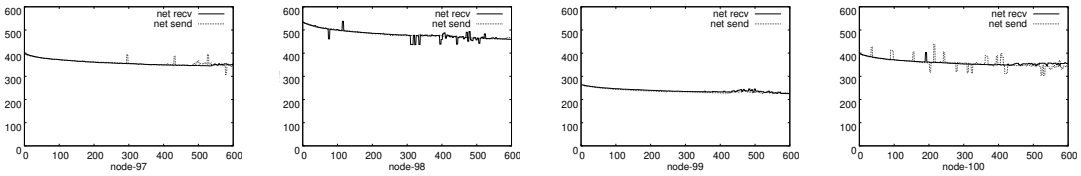

net recv
net send
node-73

net recv
net send
node-74

net recv
net send
node-75

net recv
net send
node-76

net recv
net send
node-77

net recv
net send
node-78

net recv
net send
node-79

net recv
net send
node-80

net recv
net send
node-81

ノード数：10^2


net recv
net send
node-1

net recv
net send
node-2

net recv
net send
node-3

net recv
net send
node-4

net recv
net send
node-5

net recv
net send
node-6

net recv
net send
node-7

net recv
net send
node-8

net recv
net send
node-9

net recv
net send
node-10

net recv
net send
node-11

net recv
net send
node-12

net recv
net send
node-13

net recv
net send
node-14

net recv
net send
node-15

net recv
net send
node-16

net recv
net send
node-17

net recv
net send
node-18

net recv
net send
node-19

net recv
net send
node-20

net recv
net send
node-21

net recv
net send
node-22

net recv
net send
node-23

net recv
net send
node-24

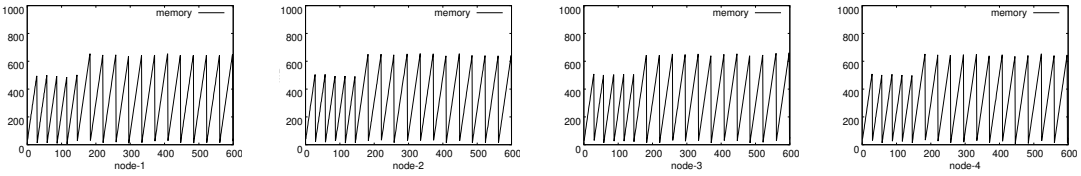

net recv
net send
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60


net recv
net send
node-61
node-62
node-63
node-64
node-65
node-66
node-67
node-68
node-69
node-70
node-71
node-72
node-73
node-74
node-75
node-76
node-77
node-78
node-79
node-80
node-81
node-82
node-83
node-84
node-85
node-86
node-87
node-88
node-89
node-90
node-91
node-92
node-93
node-94
node-95
node-96

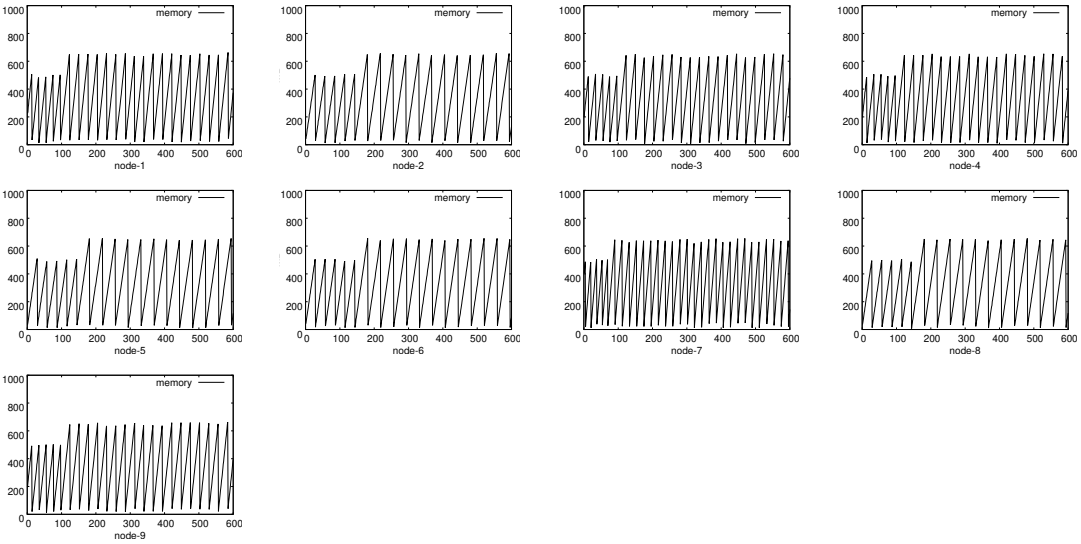


A.1.3 メモリ利用量（MB）

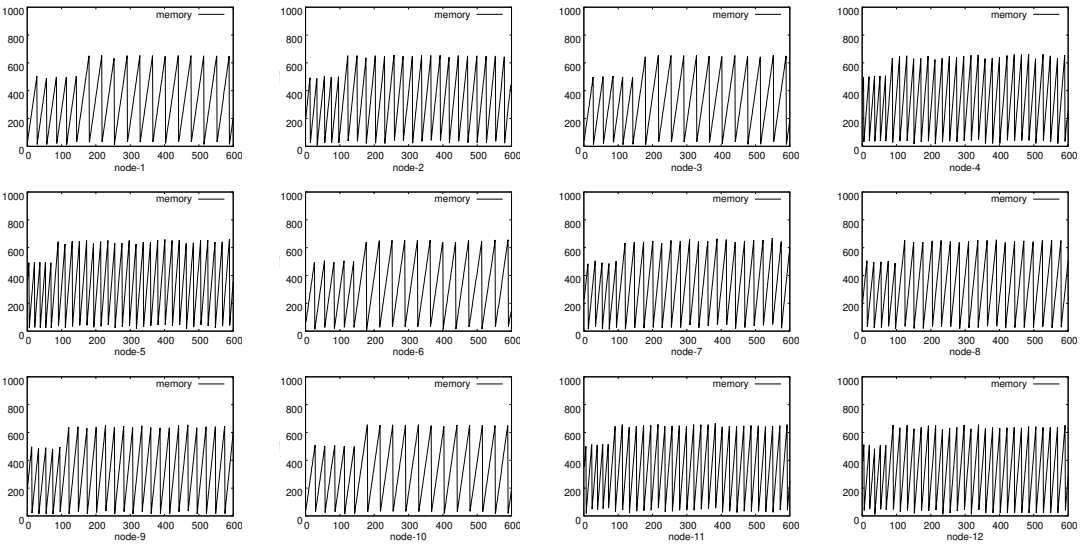
ノード数：2^2

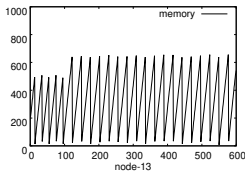


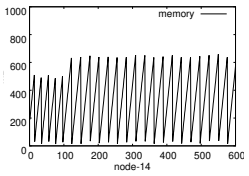
ノード数：3^2

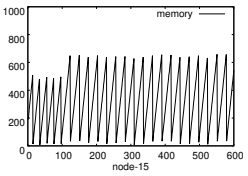


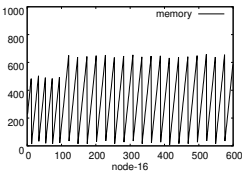
ノード数：4^2



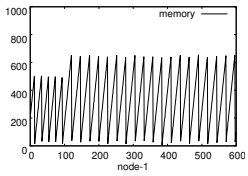


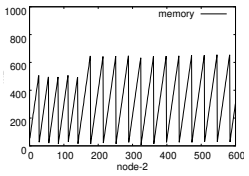


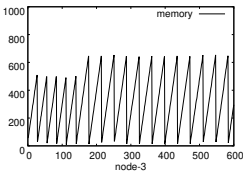


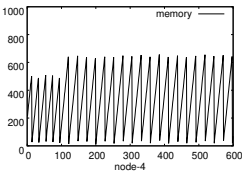


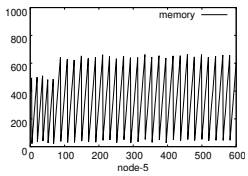
ノード数：5^2

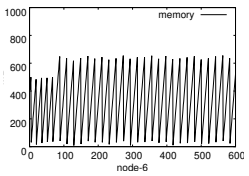


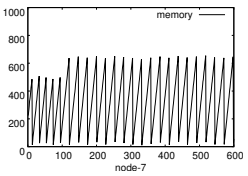


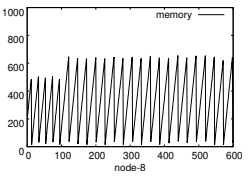


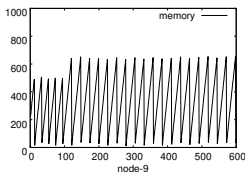


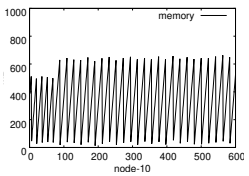


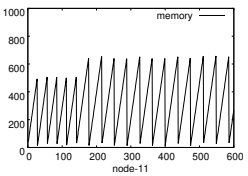


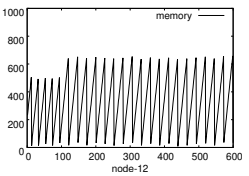


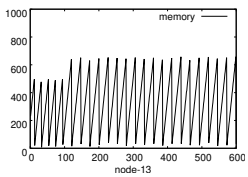


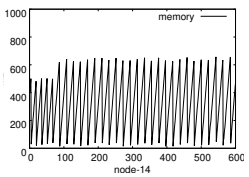


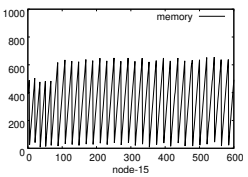


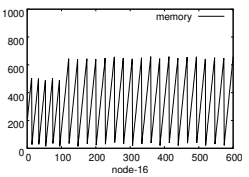


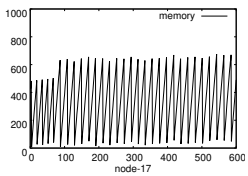


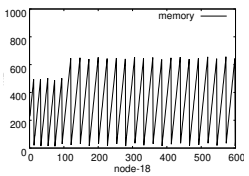


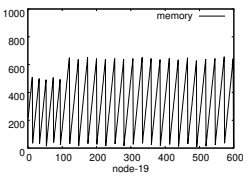


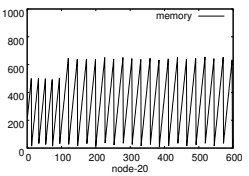


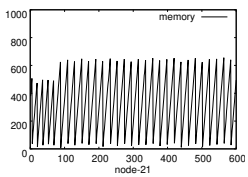


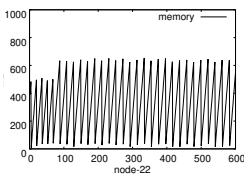


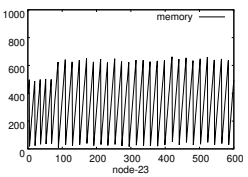


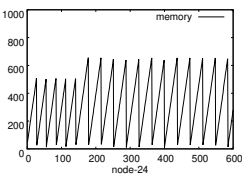


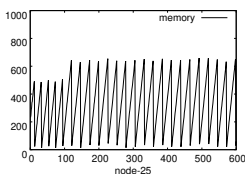




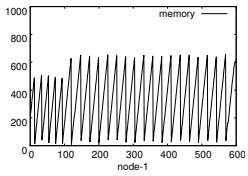


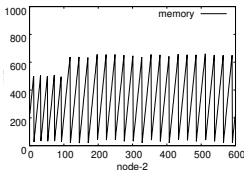


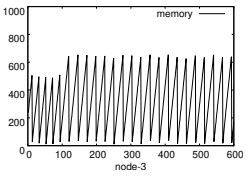


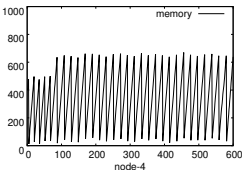


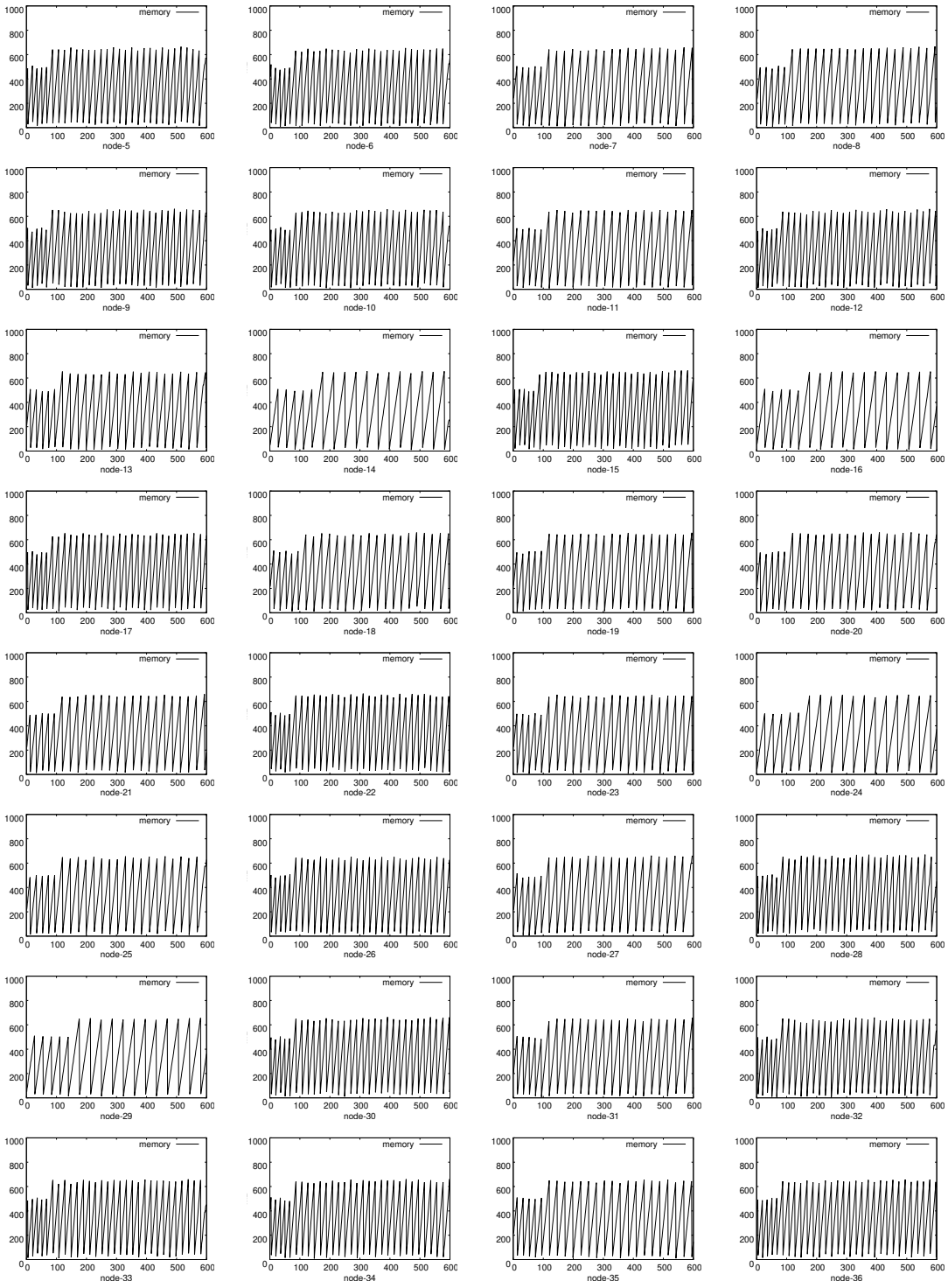
ノード数：6^2



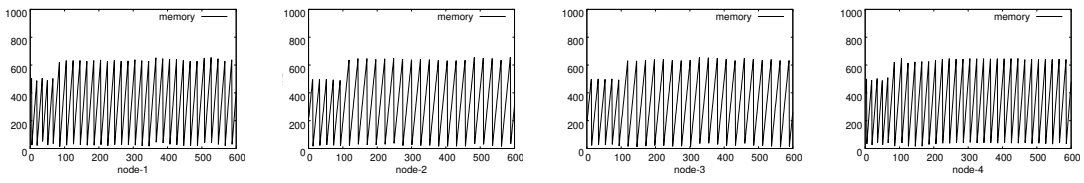


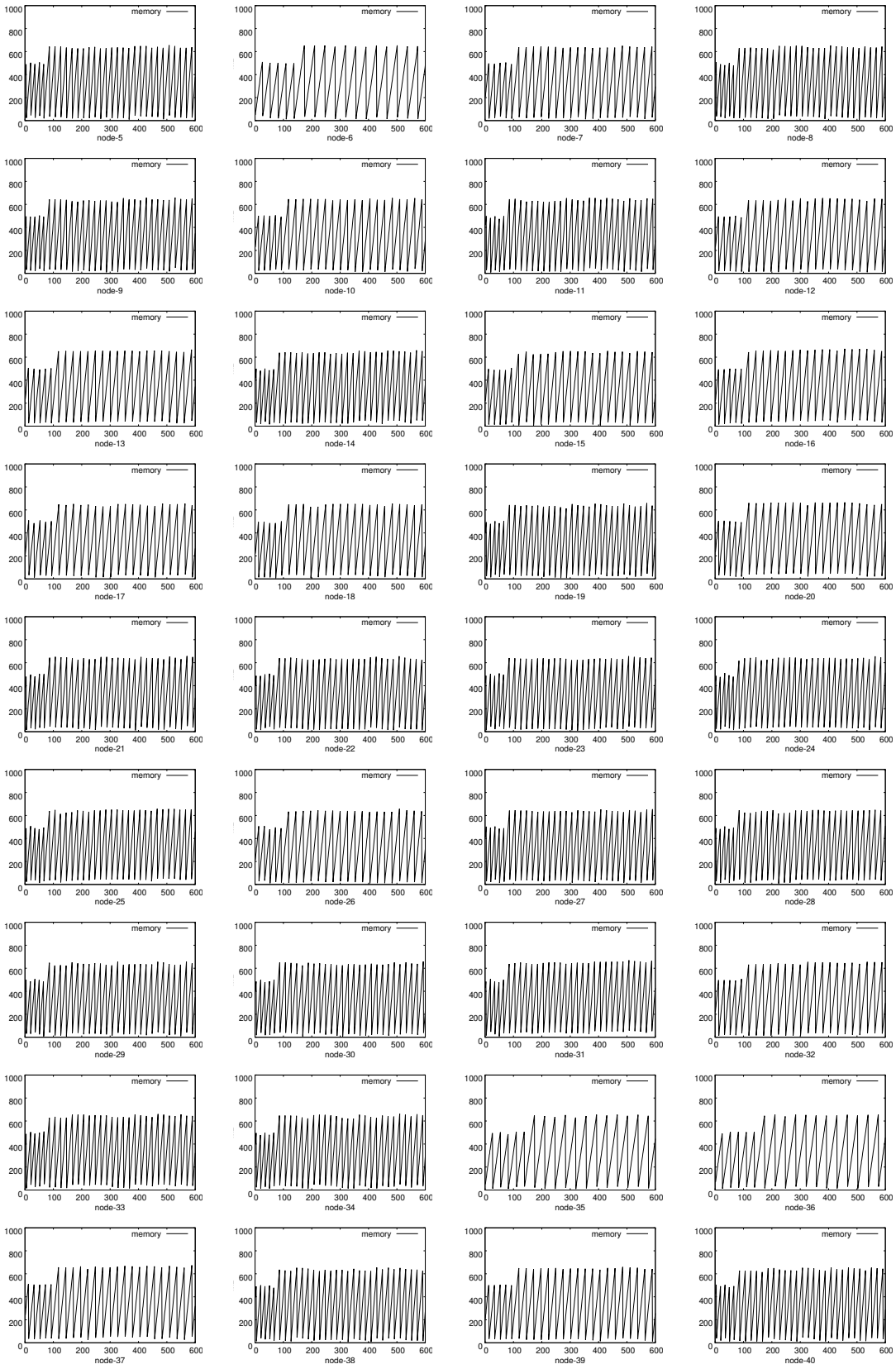


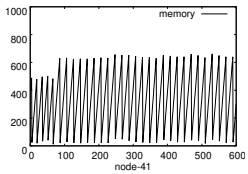
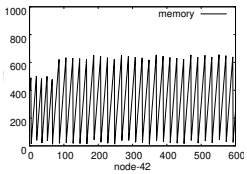
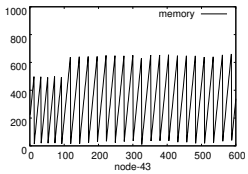
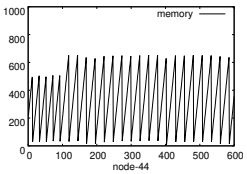
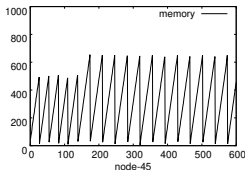
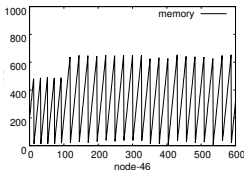
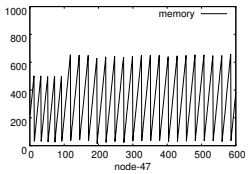
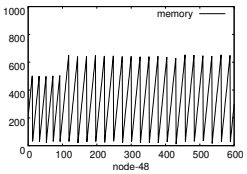
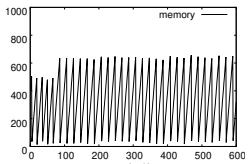



memory
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36

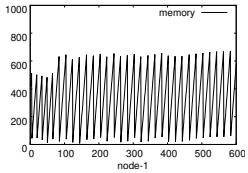
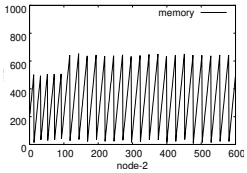
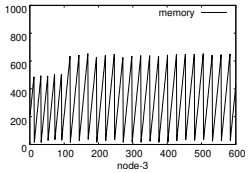
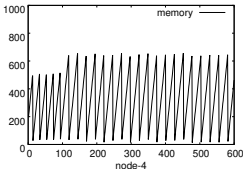
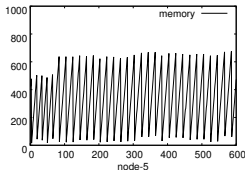
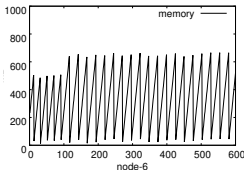
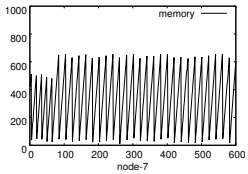
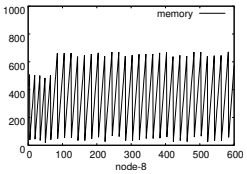
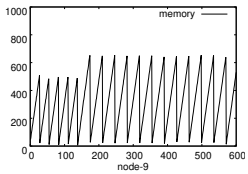
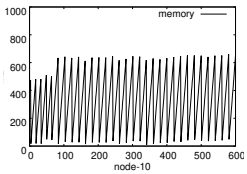
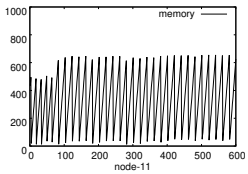
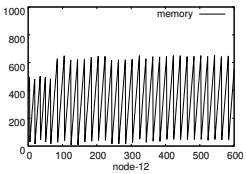
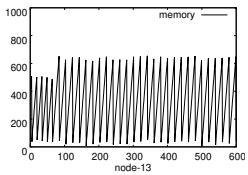
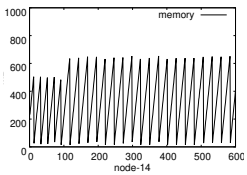
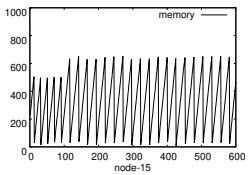
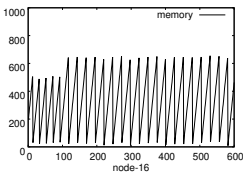
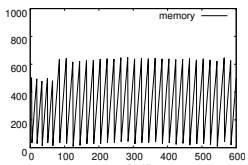
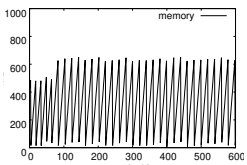
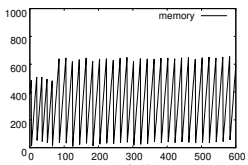
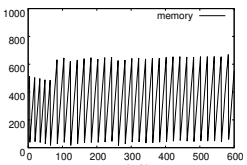
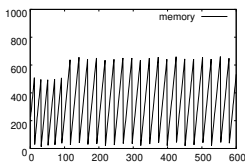
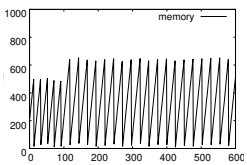
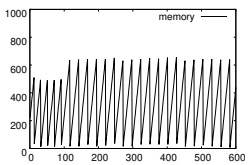
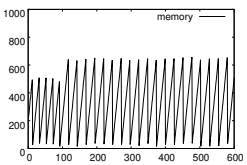
ノード数：7^2

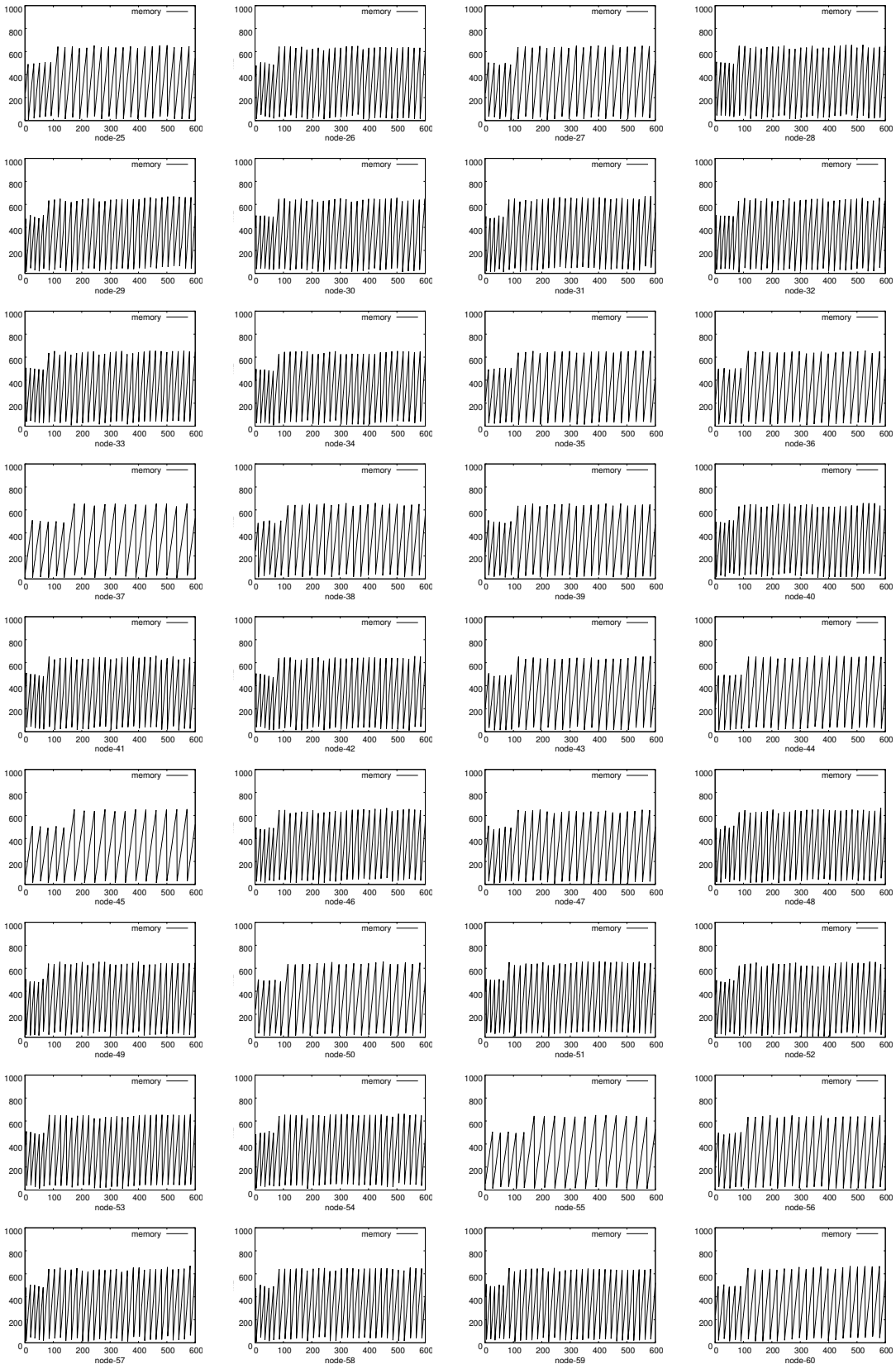

memory
node-1
node-2
node-3
node-4

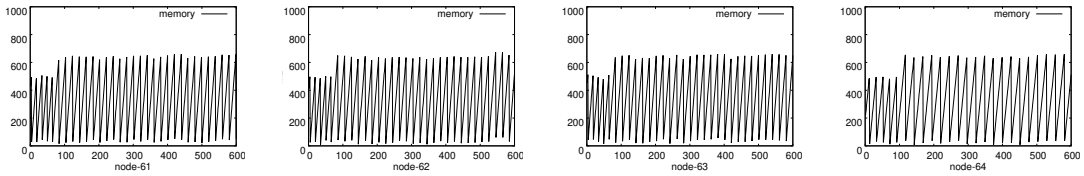

memory
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40


memory
node-41

memory
node-42

memory
node-43

memory
node-44

memory
node-45

memory
node-46

memory
node-47

memory
node-48

memory
node-49

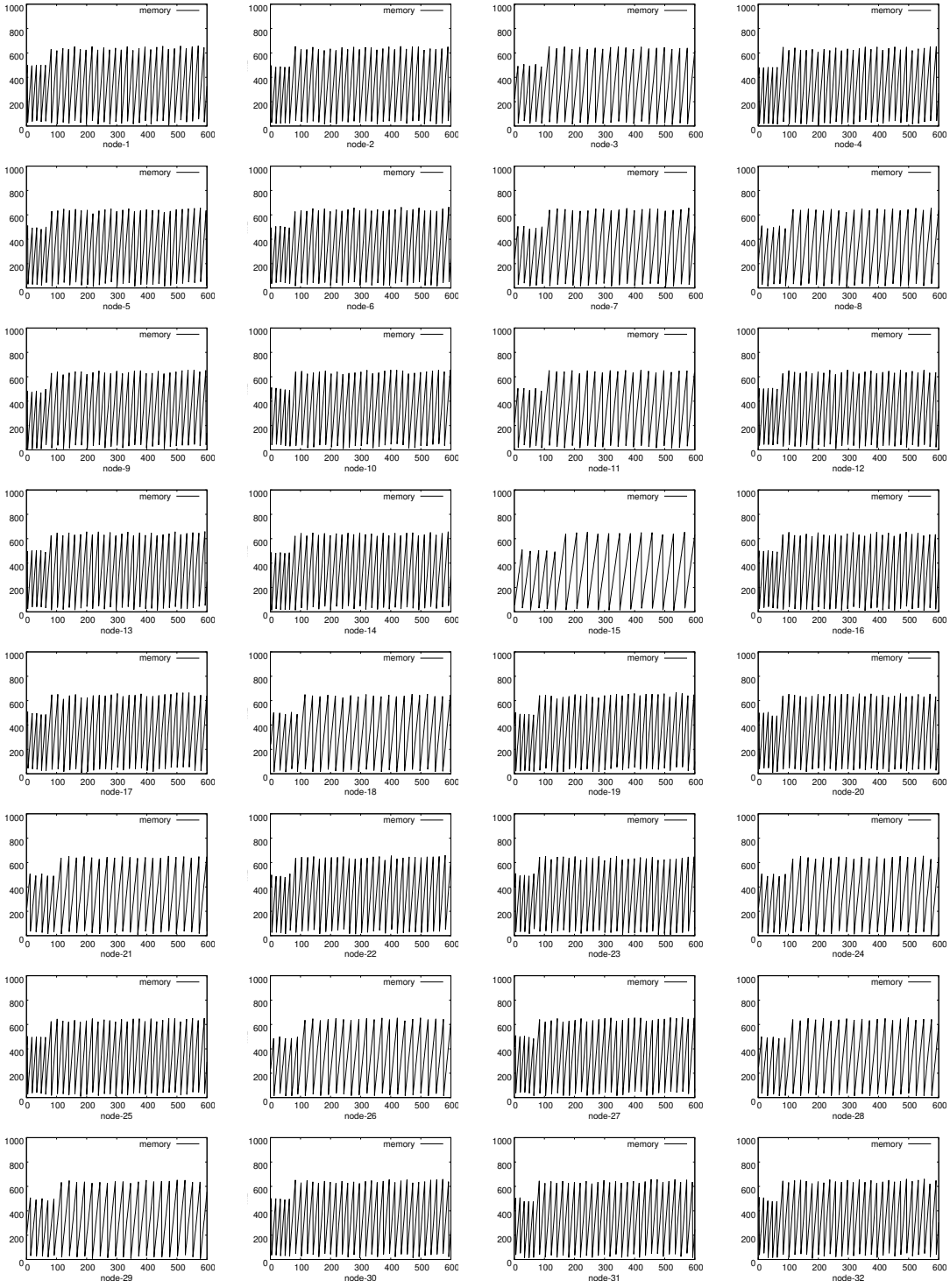
ノード数：8^2

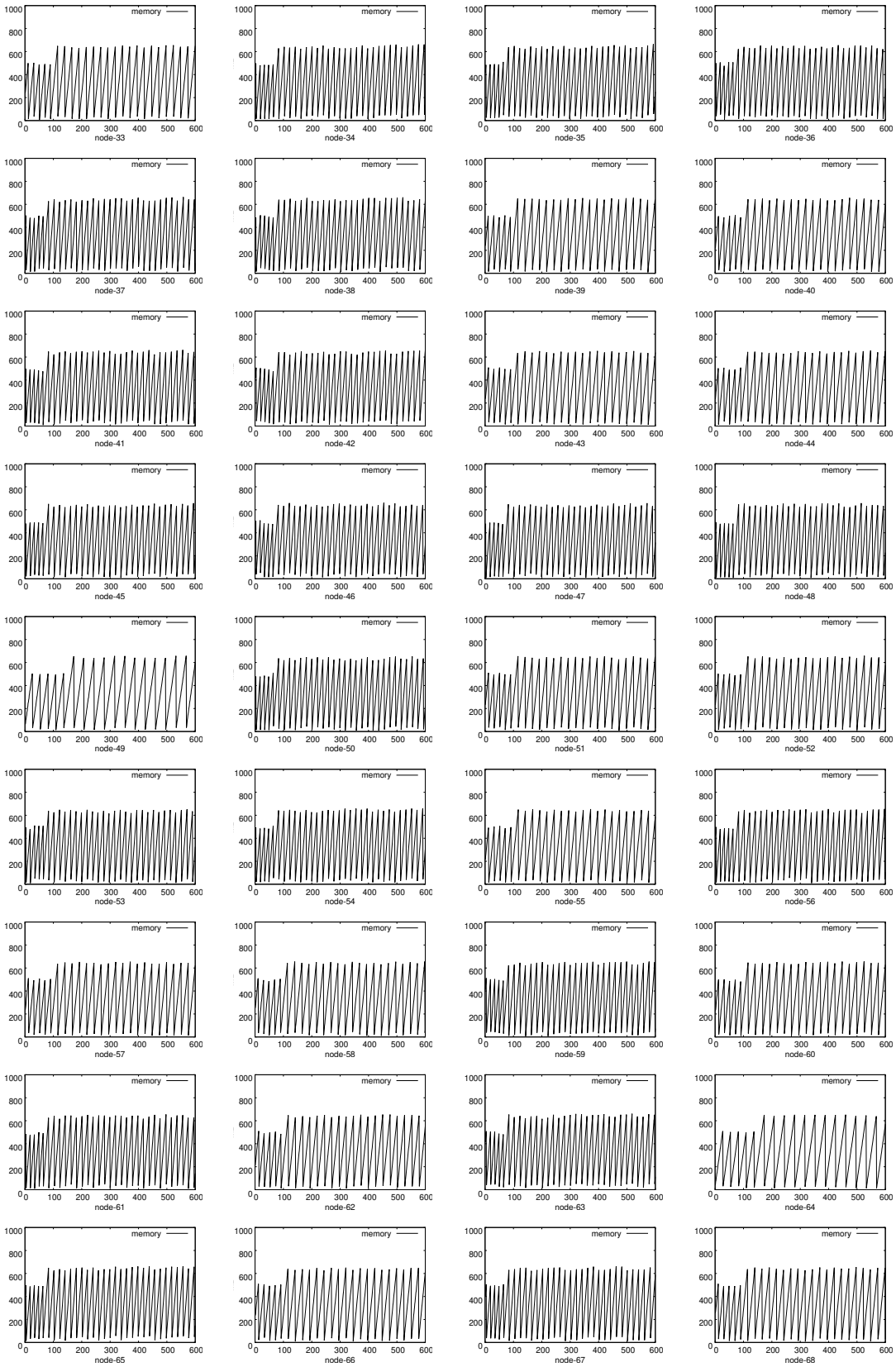

memory
node-1

memory
node-2

memory
node-3

memory
node-4

memory
node-5

memory
node-6

memory
node-7

memory
node-8

memory
node-9

memory
node-10

memory
node-11

memory
node-12

memory
node-13

memory
node-14

memory
node-15

memory
node-16

memory
node-17

memory
node-18

memory
node-19

memory
node-20

memory
node-21

memory
node-22

memory
node-23

memory
node-24

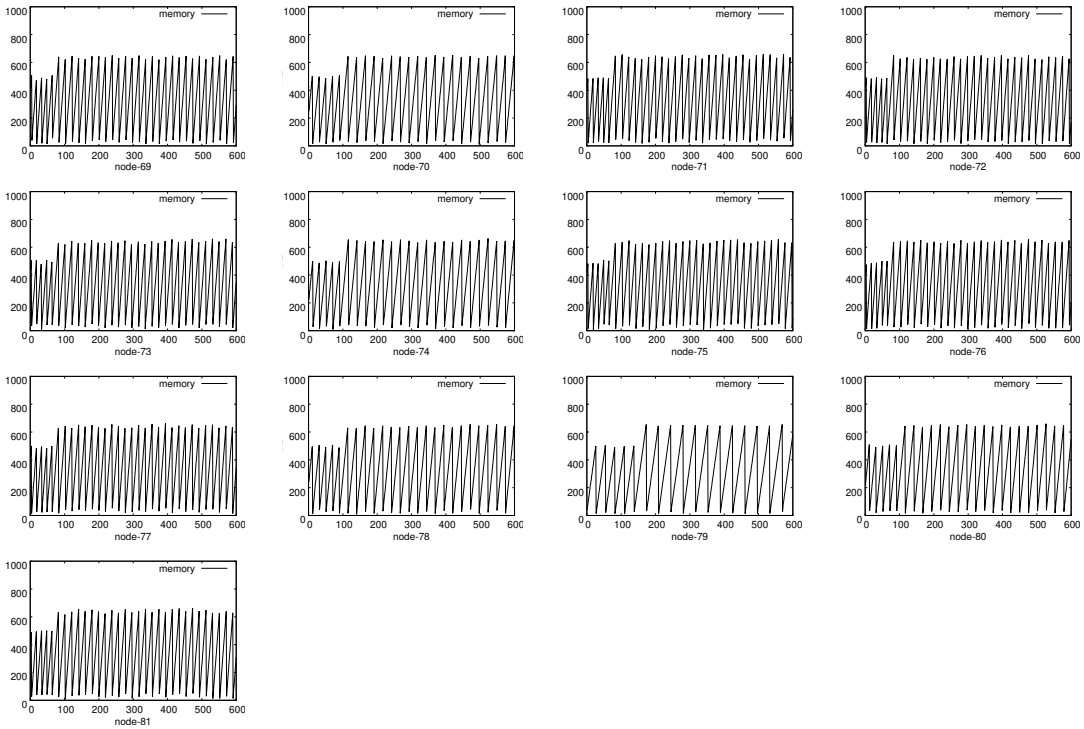

memory
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60


memory
node-61
node-62
node-63
node-64

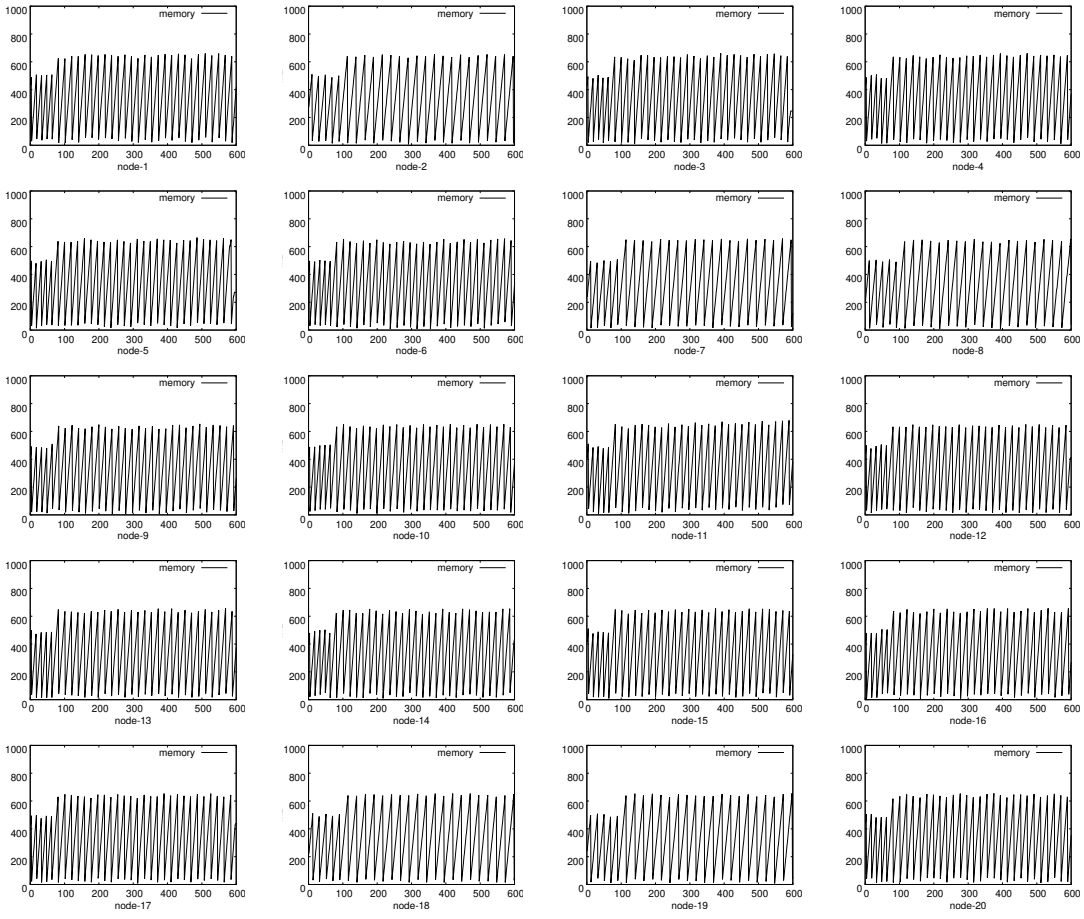
ノード数：9^2

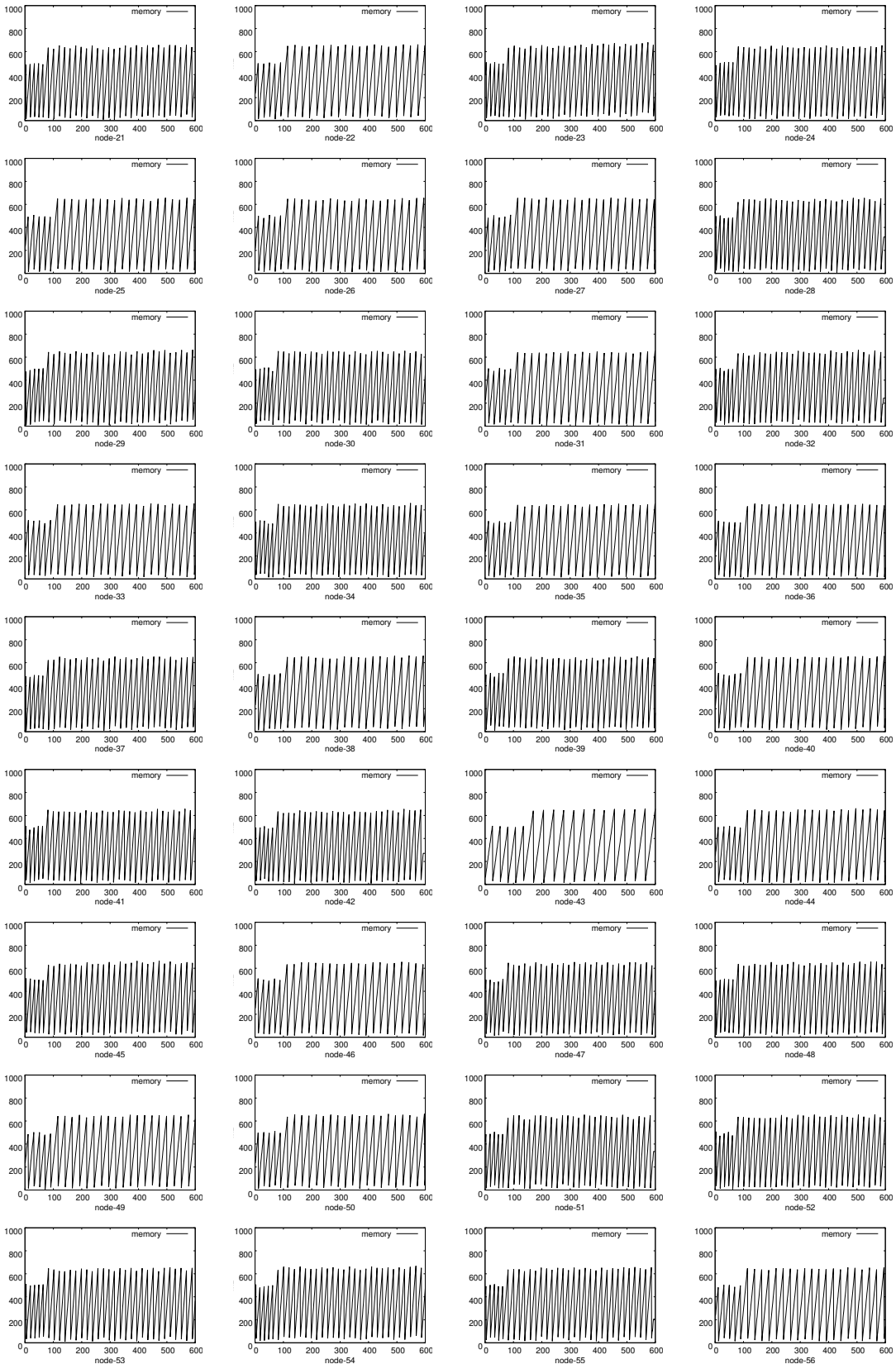

memory
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32

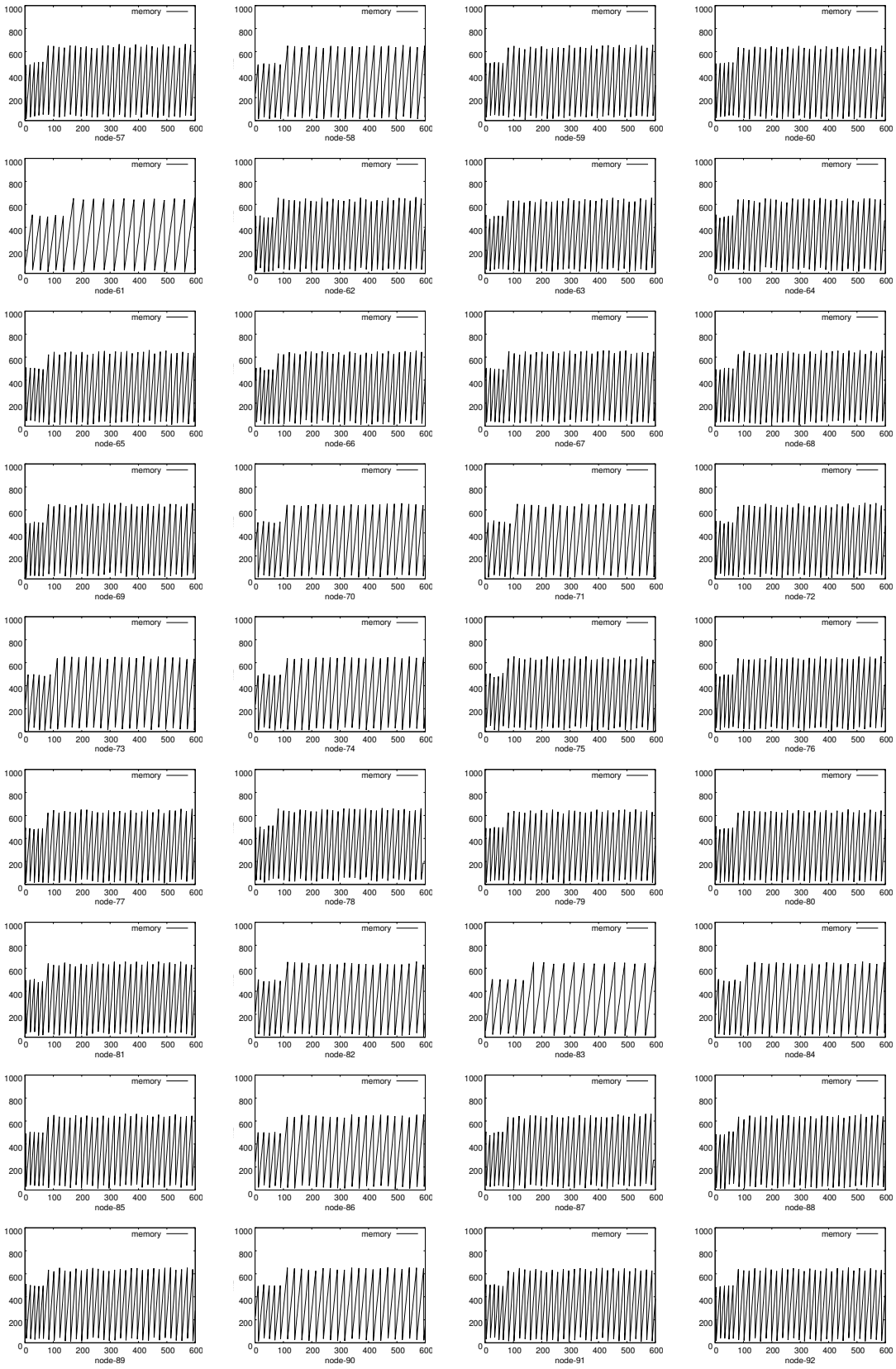

memory
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64
node-65
node-66
node-67
node-68

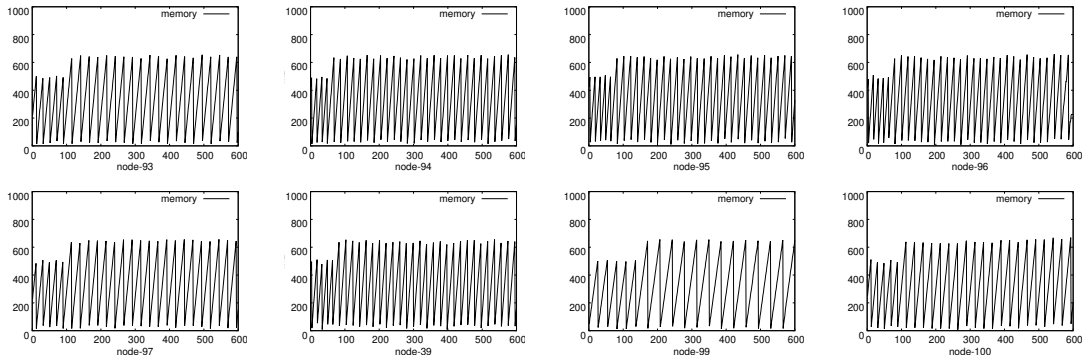

memory
node-69
node-70
node-71
node-72
node-73
node-74
node-75
node-76
node-77
node-78
node-79
node-80
node-81

ノード数：10^2


memory
node-1
node-2
node-3
node-4
node-5
node-6
node-7
node-8
node-9
node-10
node-11
node-12
node-13
node-14
node-15
node-16
node-17
node-18
node-19
node-20


memory
node-21
node-22
node-23
node-24
node-25
node-26
node-27
node-28
node-29
node-30
node-31
node-32
node-33
node-34
node-35
node-36
node-37
node-38
node-39
node-40
node-41
node-42
node-43
node-44
node-45
node-46
node-47
node-48
node-49
node-50
node-51
node-52
node-53
node-54
node-55
node-56


memory
node-57
node-58
node-59
node-60
node-61
node-62
node-63
node-64
node-65
node-66
node-67
node-68
node-69
node-70
node-71
node-72
node-73
node-74
node-75
node-76
node-77
node-78
node-79
node-80
node-81
node-82
node-83
node-84
node-85
node-86
node-87
node-88
node-89
node-90
node-91
node-92



A.2 ノードのネットワーク参加方式

3 章で述べた基盤は，サービスを立ち上げる時点では少数のノードによる小さな仮想世界から運用を開始し，ユーザの増加とともにノードを逐次追加することで，仮想世界の面積を漸増させていく運用形態を想定している．なお，参加したノードのサブワールドはノード自身が何かの情報を参照することなく初期化できるものとする．例えばゲームであれば，固定値による一様な平野であったり，あるいは地形生成アルゴリズムによって自動生成した地形などによって初期化することを想定している．格子状に接続されたノード群に対して新規ノードが参加する方法としては大きく分けて次の 2 つの方式が考えられる．

管理サーバ方式

各ノードの IP アドレスおよび，ノード間の接続関係を保持する代表サーバを稼働させる．ノードはクラスタへ参加する際に，代表サーバへアクセスし，自らが接続すべきノードの IP アドレスと接続方向（東西南北）をノードから受け取る．ノードは受け取った情報をもとに他のノードへ接続し，接続が完了したらその旨を参加完了通知として代表サーバに通知する．代表サーバは参加完了通知を受信すると，新規参入ノードの IP アドレスを記録しノード間の接続関係情報を更新する．代表サーバはワールドの形状がなるべく等方的に拡がっていくように，あるいは過去に参加したノードの離脱に伴って生じたワールド内の欠落領域を埋めるようにノードの接続位置を決定する．これらのシーケンスを図 A.1 に示す．

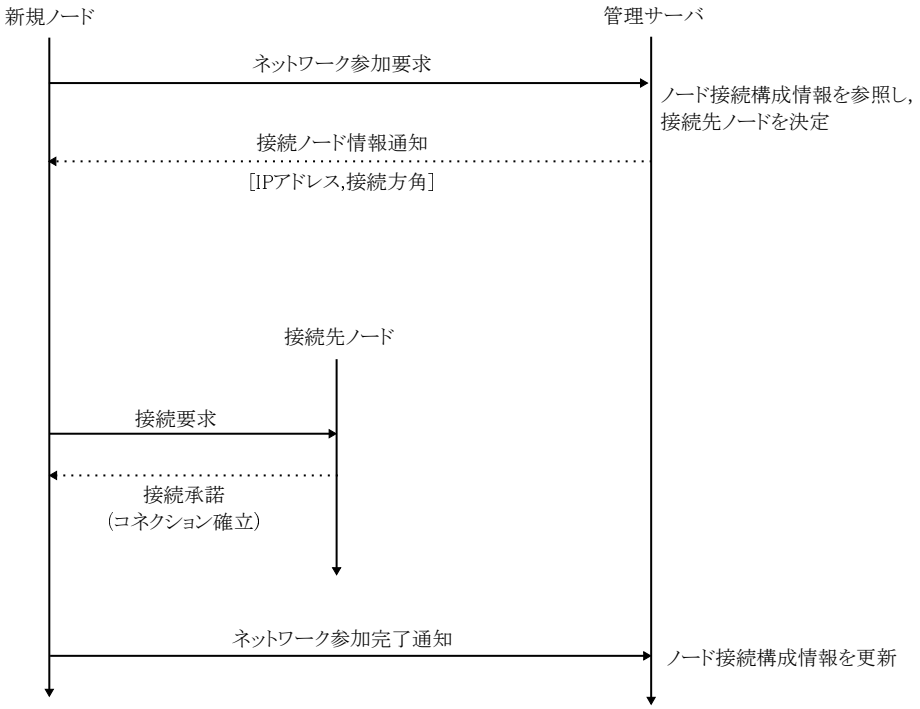


図 A.1: 管理サーバ方式によるノードのネットワーク参加シーケンス

P2P 方式

各ノードが自ノードの接続先情報を定期的に相互共有することで，ノード間の接続関係データを生成して保持しておく．ノードはクラスタへ参加する際に，クラスタ内のいずれかのノードにアクセスし，そのノードから提供される接続関係に基づき，自ら接続ノードを決定して接続する．接続後は定期的に接続先情報を周囲のノードへ送信する．初回にアクセスするノードについては別途，幾つかのノードアドレスが記されたリストが提供されるものとし，その中から一つのノードを選択してアクセスする．接続先情報は，前節で示した方法で拡散されるサブワールドデータ（d_{min}）に付随させて送信する．このため，ノードが保持する接続関係情報は，そのノードを中心として，式 3.12 で $k = d_{max}$ と置いて得られる拡散半径の距離範囲内のノードに関するものに限定される．もし，ノードが初回アクセス時に得られた接続関係情報から，接続できるノードが発見できなかった場合は，その接続関係情報に含まれる他のノード（なるべく初回にアクセスしたノードとは距離が離れているノードが望ましい）に対して再度接続関係情報をリクエストする．これらのシーケンスを図 A.2,A.3A.4 に示す．

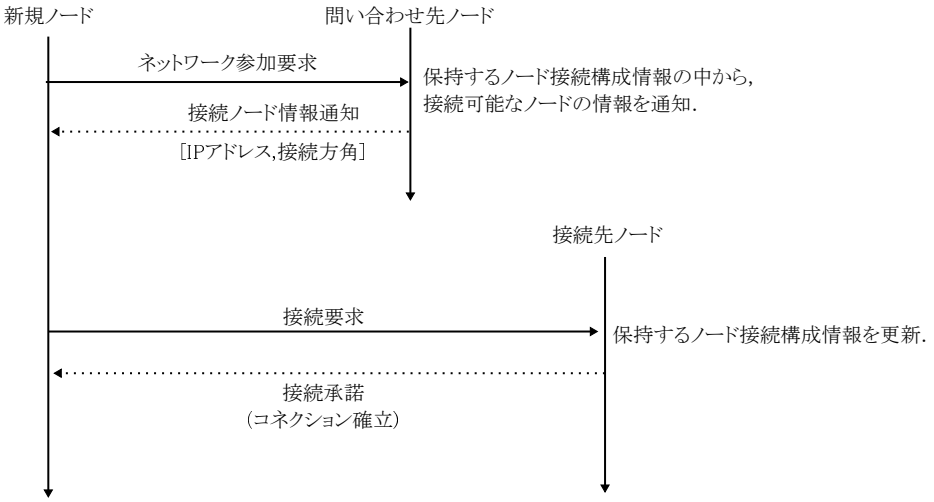


図 A.2: P2P 方式によるノードのネットワーク参加シーケンス

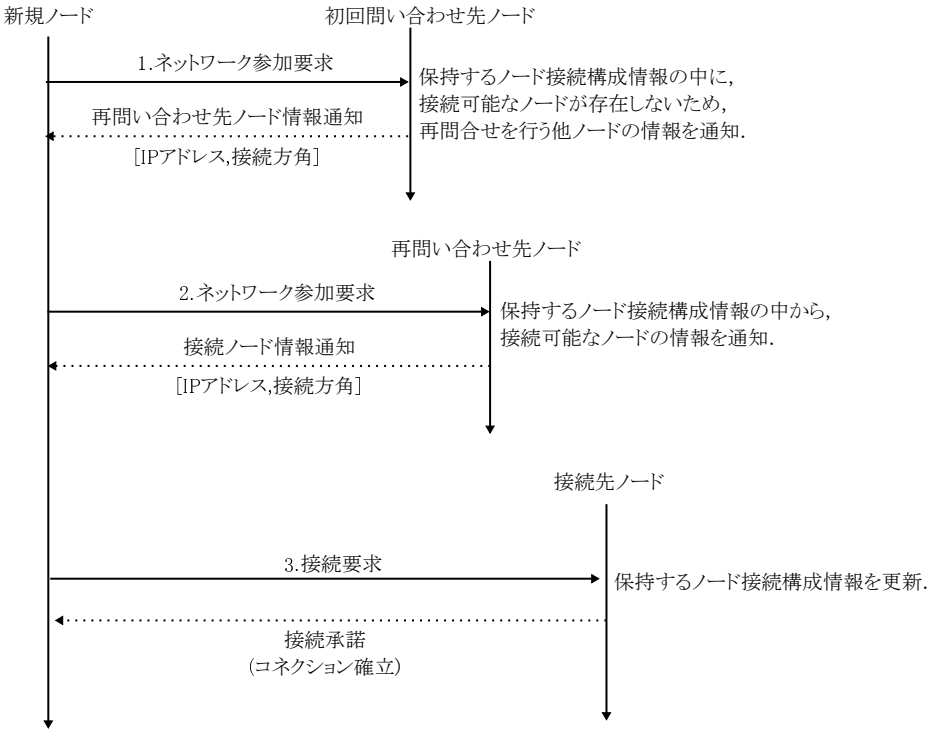


図 A.3: P2P 方式によるノードのネットワーク参加シーケンス（ネットワークへの参加要求を再試行するケース）

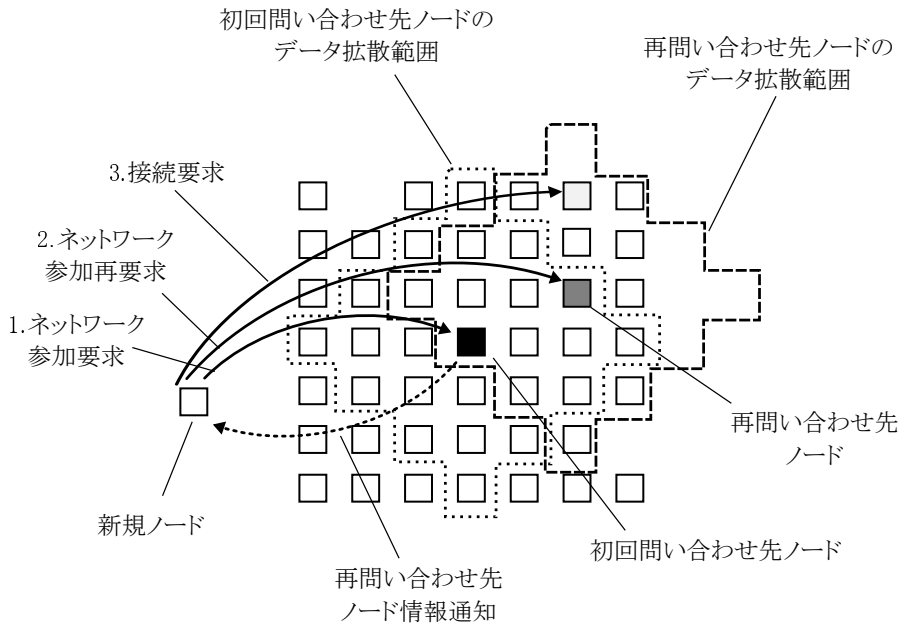


図 A.4: P2P 方式によるノードのネットワーク参加の流れ（ネットワークへの参加要求を再試行するケース．図 A.3 に対応.）

A.3　クライアントにおけるノードアドレスの解決方法

3 章で述べたシステムのクライアントが，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，その位置のサブワールド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アドレスサーバを用意することが考えられる．ただしこの場合はアドレス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．この方式のシーケンスを図 A.2 に示す．

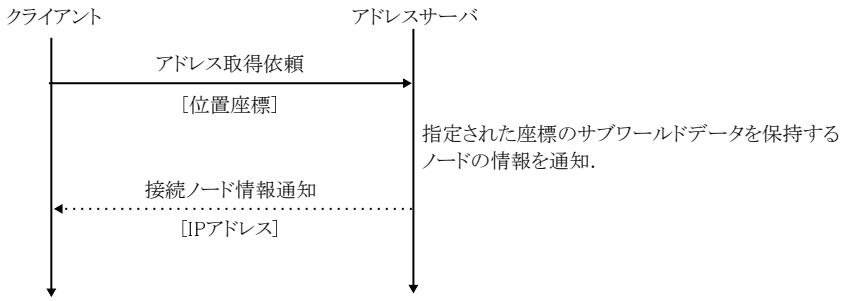


図 A.5: 集中保持方式によるクライアントのノードアドレス解決シーケンス

分散保持方式

ノードが，周囲に拡散するセ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，サブワールド情報に付与されたアドレスを辿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幾つか保持しておき，そこからランダムに選択したノードから取得するセル情報に含まれるアドレスを辿ってアクセス先を切り替える．ただし，a_{swt} 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セ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．この方式のシーケンスを図 A.6,A.7 に示す．

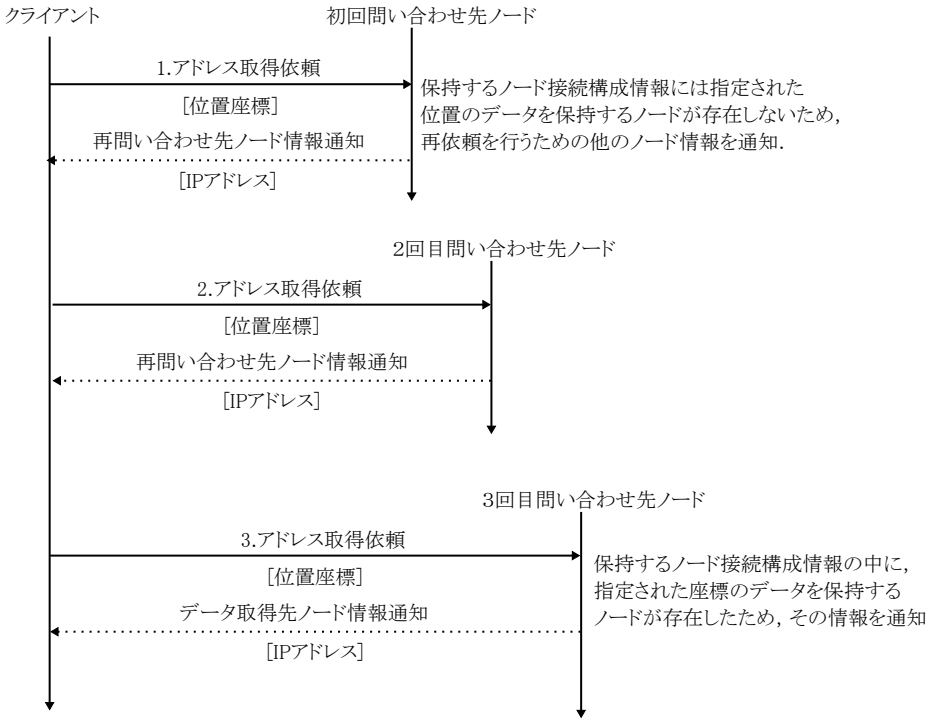


図 A.6: 分散保持方式によるクライアントのノードアドレス解決シーケンス

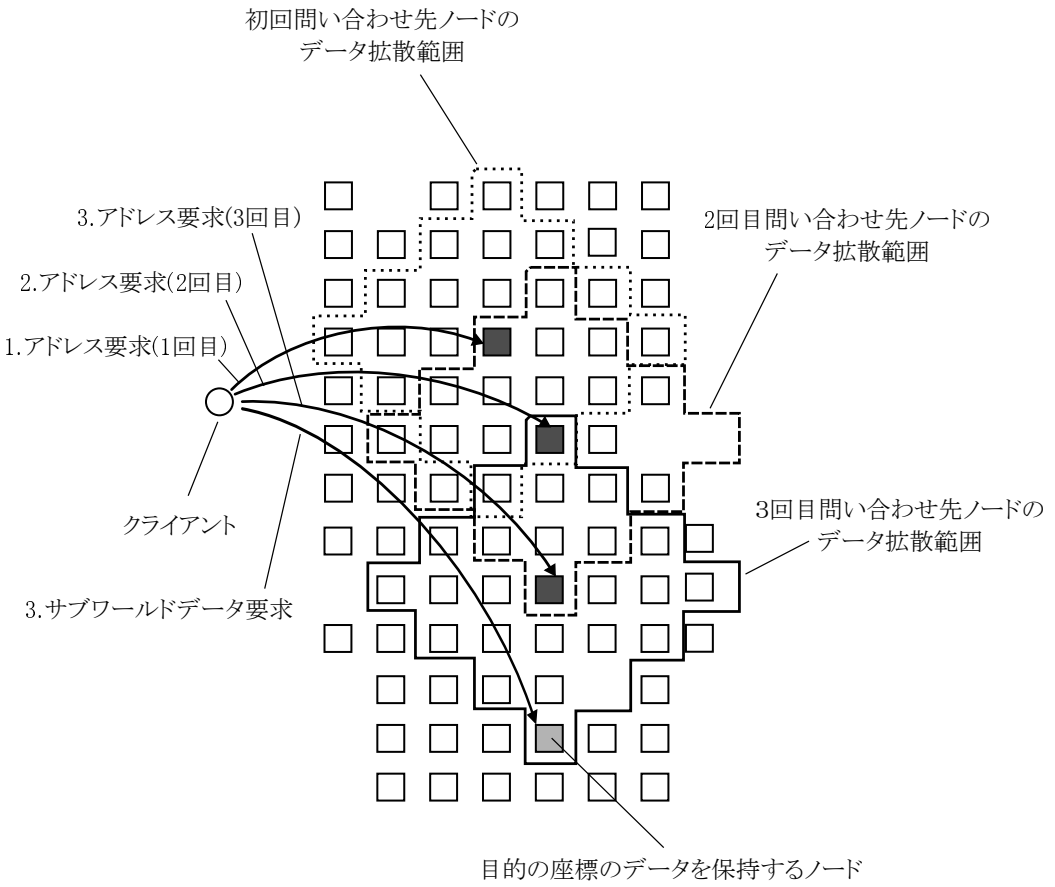


図 A.7: 分散保持方式によるクライアントのノードアドレス解決の流れ（図 A.6 に対応）

A.4　空間の全域可視化のためのデータ拡散（別方式）

ここでは，3 章で述べた方式とは異なる方法でデータを拡散する方法について述べる．3 章で述べた方式は，各ノードが LOD ピラミッドを生成し，それに LOD に応じた拡散半径を付与してデータを周辺ノードへ伝播する．この拡散半径は LOD 切替セル数 a_{swt} によって上限が規定されるため，各ノードはノードネットワーク上での周辺の限定された範囲の情報を持つことになる．この性質は，**??**章でも述べるように，空間の面積拡大に対してスケーラビリティを有する一方で，各ノードは空間全体を俯瞰する情報を持たず，ユーザが空間を俯瞰できる領域の広さは限定される．この関係を，円周上に接続した 1 次元のノード群を例にとって図 A.8 に示す．

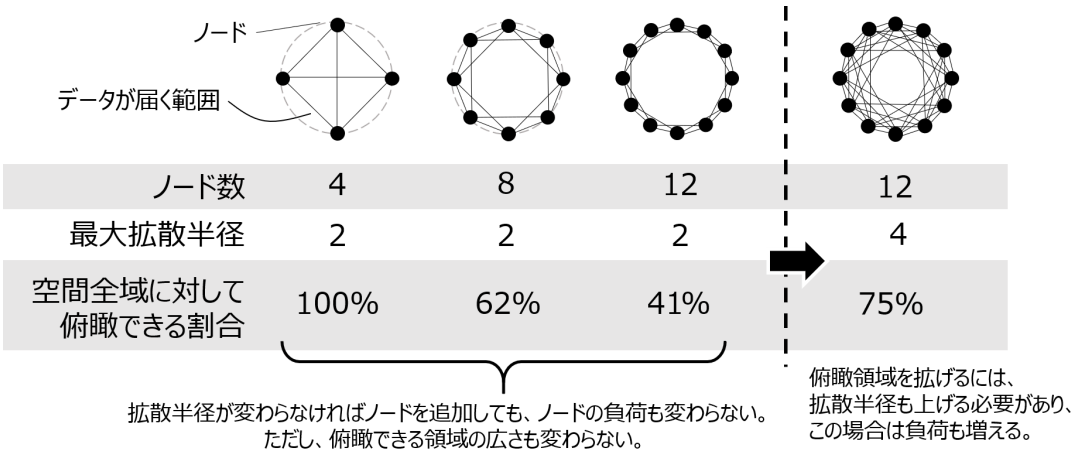


図 A.8: 拡散半径と空間全域に対して俯瞰できる割合（方式 1 ）

本章で述べる方式は，各ノードが全ノード分の情報を保持することで，ユーザが空間の限定された範囲の領域ではなく，空間全域を可視化することを可能にするものである．

A.4.1 拡散を通じたデータの疎視化

前章で述べた方式では，各ノードは保持するサブワールドをから段階的に LOD を落としたデータピラミッドを自ら生成し，それを近傍ノードへ送信するものであった．これに対し，本章での方式は，各ノードが自らのサブワールドを疎視化するデータを生成するのではなく，サブワールドデータをそのまま拡散し，拡散の過程を通じて経由するノードにおけるサブワールドデータを含めながら疎視化していくことで，最終的には全てのノードの情報が含まれた疎視化データを生成するものである．

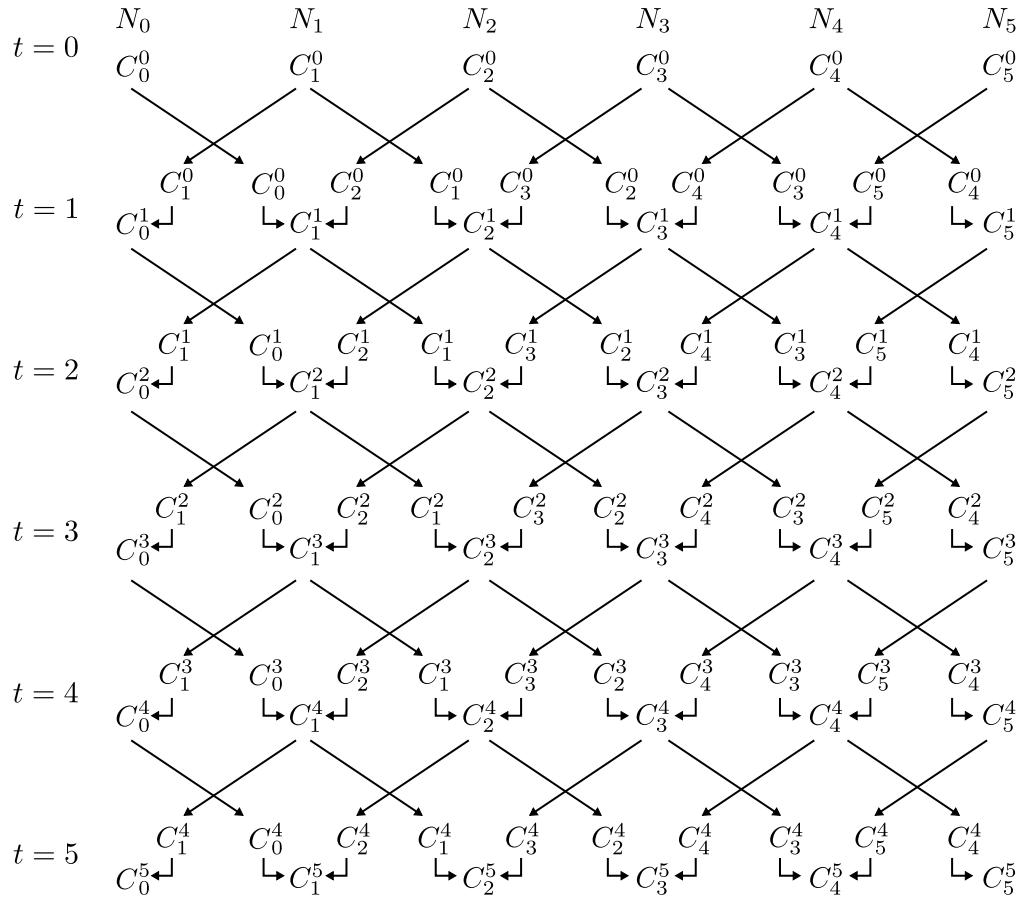


図 A.9: 拡散を通じたデータ疎視化のプロセス

説明のために 1 次元で結合した 6 台のノードを用いてこのプロセスを説明する．図 A.9 では，ノードが $N_0, N_1, \cdots, N_5$ と接続されており，$t = 0$ の時点で，それぞれがサブワールド $C_0^0, C_1^0, \cdots, C_5^0$ を保持している．まず時刻 $t = 1$ で，各ノードは近接している左右のノードに対して自らが保持するサブワールドを送信する．結果としては各ノードは自らの分と，左右のノードから送信されてきた分と併せて，合計 3 つのサブワールドを保持することになる．次にこの 3 つのサブワールドを 1 つに縮約する処理を行う．具体的には図 A.10 に示すように，それぞれのサブワールドの長さを 1/3 に縮小した上で連結する（今はサブワールドも 1 次元と想定している）．結果として縮約後のサブワールドのサイズは，もとのサブワールドと同一となる．この縮約後のサブワールドを $C_0^1, C_1^1, \cdots, C_5^1$ と表現する．これを縮約回数 1 のサブワールドデータと呼ぶ．次に $t = 2$ で，各ノードが保持している縮約後のサブワールドをさらに近接しているノードに対して送信する．そして，各ノードは同じように 3 つのサブワールドを 1 つに縮約する．これを $C_0^2, C_1^2, \cdots, C_5^2$ とする．これを時間ステップ毎に繰り返していくと，$t = 5$ の時点で，各ノードが保持するサブワールド $C_0^5, C_1^5, \cdots, C_5^5$

それぞれには，全てのノードの情報が含まれることになる．また，この時，ノード N_i は，$C_i^0, C_i^1, \cdots, C_i^5$ と，縮約回数が 1 ～ 5 までのサブワールドデータを保持することになる．縮約回数は，前章における LOD と同じ意味を持ち，縮約回数が多いほど，より広い範囲を俯瞰するデータである．したがって各ノードは，自ノードを中心とした局所的な詳細な情報から，全体を俯瞰する簡略化された情報までを段階的に保持することになり，これをユーザからの要求に応じて提供することで，ユーザは複数のノードにアクセスすることなく，俯瞰的な情報と局所的な情報の両方を得ることができる，

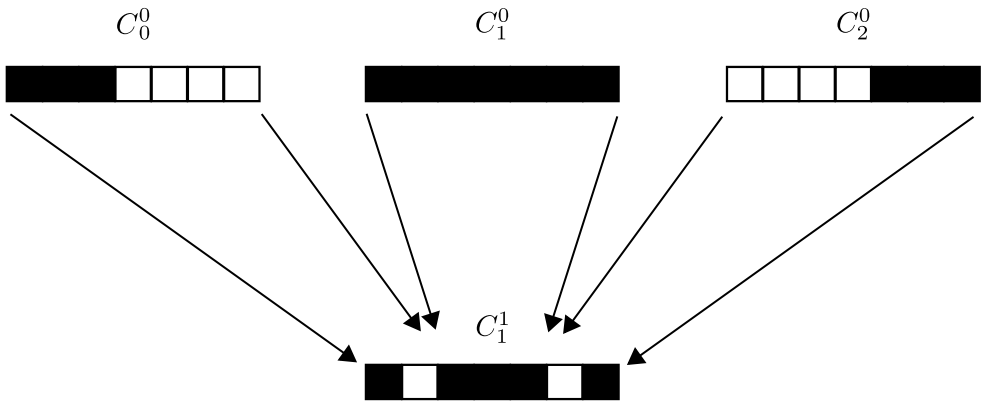


図 A.10: データの縮約の様子（ 1 次元の場合）

図 A.12 に，5×5 の格子状に接続した合計 25 ノードが，上記の手順に従って，それぞれのノードが保持するサブワールドデータを拡散した結果を示す．各ノードが，それぞれに割り当てられた縮約されていない元のサブワールドデータ，段階的に周囲の広い範囲のノードのサブワールドデータに加え，全てのノード分の情報が含まれたサブワールドデータを保持していることがわかる．なお，上記手順では空間の次元が 1 次元という仮定であったため， 1 回のデータの伝播で各ノードは左右の近接ノードからの情報を受信し，それを縮約できた．これに対し，空間が 2 次元である場合， 1 回の縮約のためには 2 回のデータ伝播が必要となる（図 A.11）

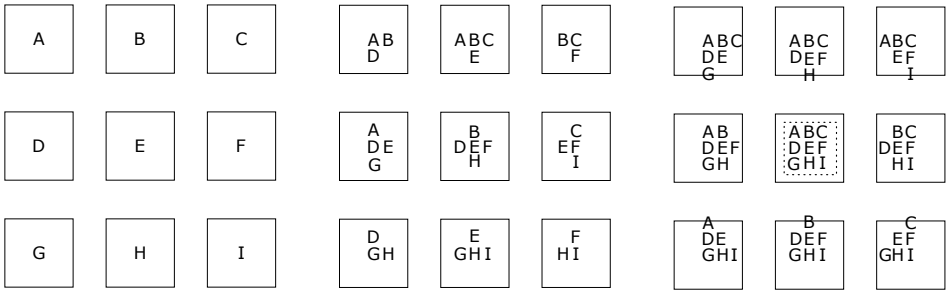


図 A.11: データの縮約の様子（２次元の場合）：点線の矩形が縮約対象

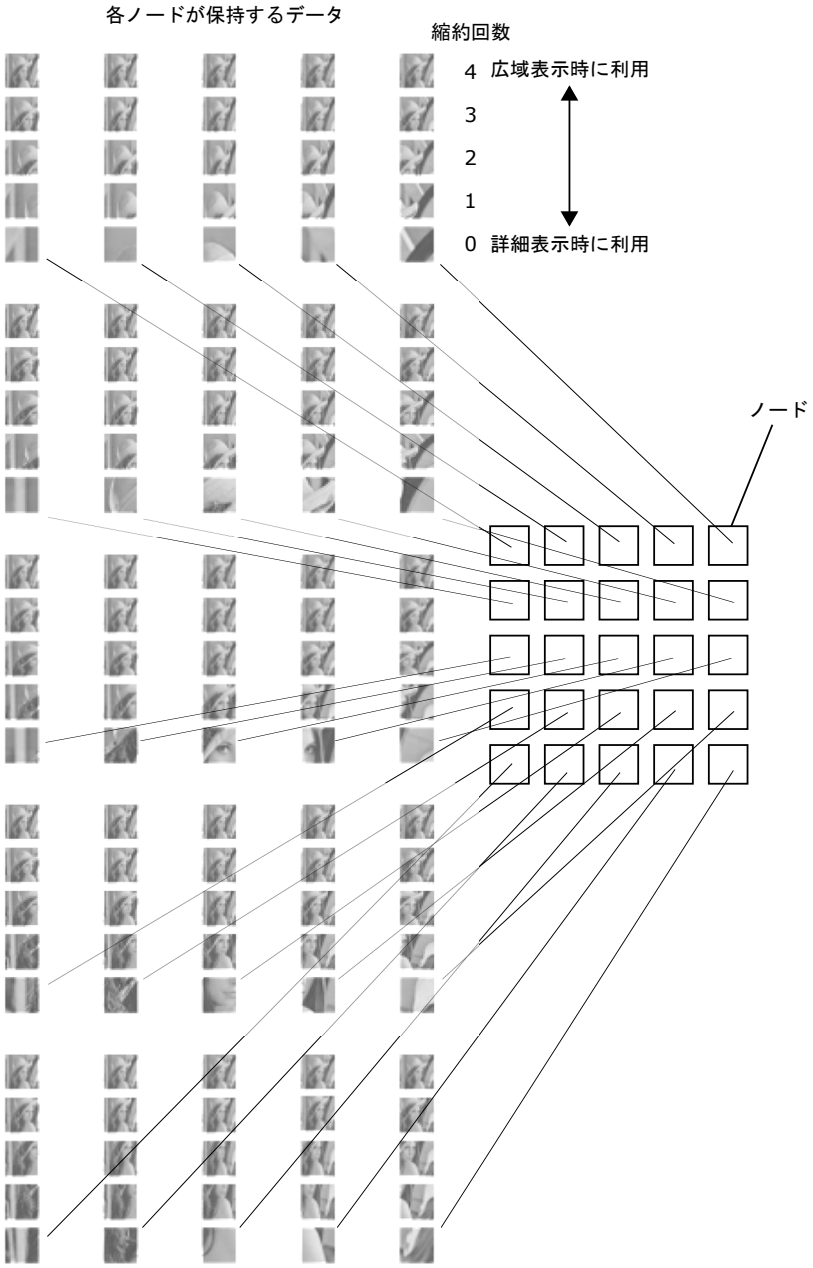


図 A.12: ノードが保持する縮約データの例

A.4.2　疎視化レベルに応じたデータの転送周期調整

前節で述べた方式では，ノードの全体数が増加すると，各ノードに他のノードから送信されてくるデータの量も多くなり，その処理やネットワークの帯域にかかる負荷が大きくなる．このままでは，ノード全体数の増加に対するスケーラビリティを有さないため，本節ではこれへの対応方式を述べる．

前節で述べた方式では，各ノードに送信されてくるサブワールドの数が時間ステップが経過するにつれて増加する．例えばノード数 n を 1 次元で接続した状況では，各ノードに対して，時間ステップ t の時に $2t$ のデータが送信され，最終的には時間ステップ毎に $n-1$ 個の縮約されたサブワールドデータが送信されてくる（図 A.13）．したがって，各ノードに対する負荷の量は $O(n)$ となり，ノードの負荷を一定以下に保つためには全体ノード数が大きく制限されてしまう．

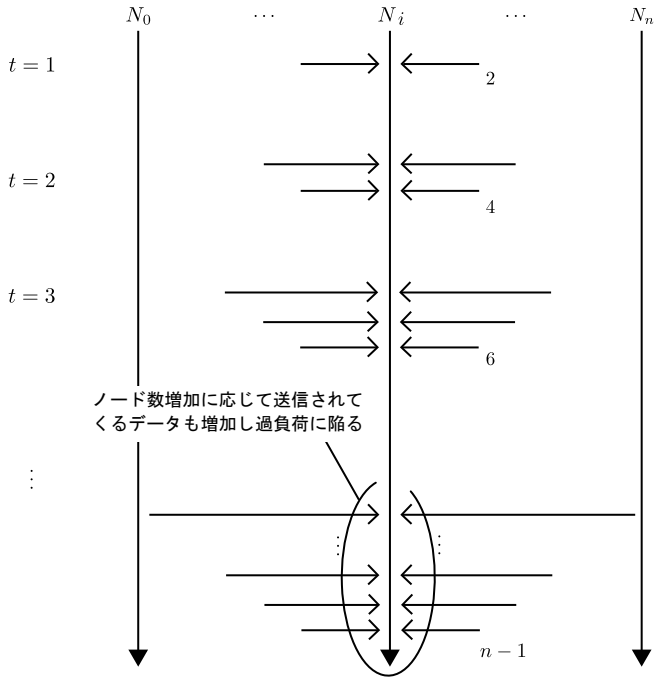


図 A.13: ノードに対して送信されてくるデータ量

この問題に対応するため，データを送信する時間間隔を縮約回数に応じて長くとる．具体的な手順として，各ノードは，自らが保持する縮約されていない元のサブワールドデータを毎時間ステップ毎に近接ノードに対して送信する．次に，そのデータを受信したノードは，データを受信する度に毎回ではなく，データを 2 回受信する度に，その時に受信し

たデータを縮約して転送する．つまり，$2k\{k = 1, 2, 3 \cdots\}$ 回目に受信したデータを縮約転送し，$2k + 1$ 回目に受信したデータは破棄する．次に，縮約回数 1 のデータを受信したノードは，同様に，データを 2 回受信する度に縮約転送を行う．さらにその縮約回数 2 のデータを受信したノードも同様の手順を繰り返していく．図 A.14 に，ノードが受信するデータのタイミングを示す．これらによって，データの縮約回数 c に対して，その転送周期（タイムステップ）は 2^{c-1} となる．このような転送周期の調整を行うことで，ノードの全体数 n が増えても，各ノードに送信されてくるデータ量が $O(n)$ で増加することはなく，ノード数を増やすことが可能となる．しかしその一方で，遠く離れたノードのデータほど，更新される時間間隔が長くなる．すなわち本節で述べた対応は，空間データの更新間隔を一部犠牲にして，空間データ面積のスケーラビリティを確保するものである．

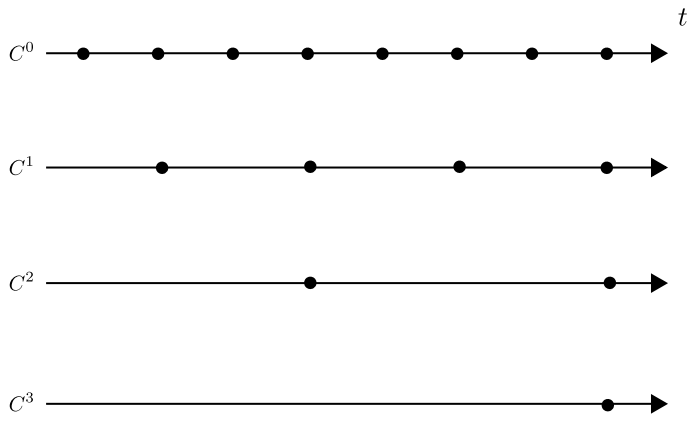


図 A.14: データの転送周期調整を行った結果，ノードがデータを受信するタイミング（データの縮約回数別）

A.4.3 全体ノード数に応じたデータの保持量調整

本節では，ノードの保持するデータ量を軽減する対処について述べる．本章で示すデータの拡散方式は，各ノードは送信されてきたデータを縮約転送するとともに，そのデータはユーザからの要求に応じてユーザに提供するために保持・更新する．この保持量がノードの全体数に応じて増加するため，このままでは，ノード全体数の増加に対するスケーラビリティ有さない．

前節で述べた方法によって，時間ステップ毎にノードが受信するデータの増加は抑えることが可能となる．しかし，時間が十分に経過するとノードが受け取る縮約回数別のデー

タ数は全体ノード数まで増加する．例えば全体ノード数 n を 1 次元に繋げた場合，図 A.15 に示すように，時間ステップ $t = 2^{k-1}\{k = 1, 2, 3 \cdots\}$ の度に新しい縮約回数のデータを，最大 $n/2$ 個まで受信することになり，ノードがこのデータを保持する量に限界があることから，全体ノード数は大きく制限されることになる．

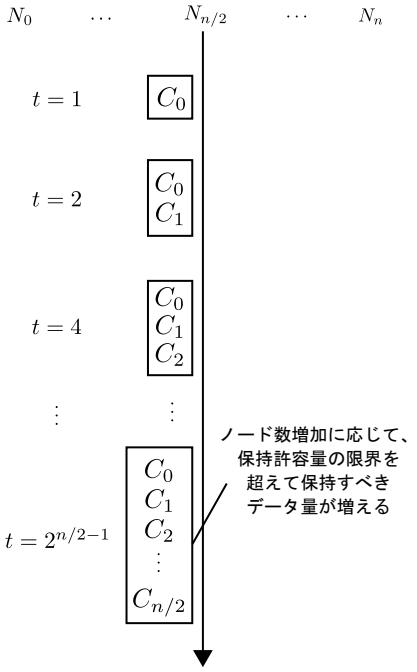


図 A.15: ノードが保持するデータ量

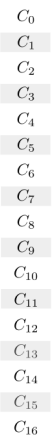


図 A.16: 全体ノード数に応じた保持データ量の削減: 網掛けしたデータが破棄対象．$n = 32, l = 8$ の場合を例にとって図示．

この問題に対応するため，各ノードがノードの全体数を考慮した上で保持するデータ量を削減する．具体的には図 A.16 に示すように，ノードが保持するデータ数の上限を l，全体ノード数を n とすると，縮約回数が $2n/l$ 回のデータ以外は保存せずに放棄し，縮約転送のみを行う．保持するデータ数の限界数 l は，ノードのメモリ搭載量に応じ事前に定めておく．また，ノードの全体数 n は，ノードが受信するデータの縮約回数から，各ノードがそれぞれ求めることができる．なお，ノードが縮約回数 $2n/l$ 以外のデータを受け取っても，データの廃棄は行ってもデータの縮約転送は実行する理由は，これを行わないとデータ拡散を通じた縮約の連鎖が継続せず，他のノードがその縮約回数以上のデータを受信できなくなるためである．この方式によって，各ノードが保持するデータの数は最大で l となる．一方で，ユーザに対して提供できるデータの縮約回数別のバリエーションは減少する．すなわち本節で述べた対応は，ユーザが空間の可視化スケールを俯瞰と詳細とで切り替える際の段階数を犠牲にすることで，空間データ面積のスケーラビリティを確保するものである．