MMORPGのユーザ生成コンテンツと親和性の高いP2Pベースの分散サーバ方式

陳 正[1]　光来 健一[1]

概要：大規模多人数同時参加型オンライン RPG（MMORPG）では，ユーザ生成コンテンツ（UGC）によるプレイヤ主導の世界構築が重要性を増している．従来の中央集権型サーバには負荷集中や拡張性の問題があり，UGC の増加によってスケーラビリティが限界に達している．そのため，分散アーキテクチャの必要が高まっているが，UGC 主導型 MMORPG を支えるには様々な課題を解決する必要がある．本稿では，MMORPG の UGC と親和性の高い，P2P を用いた分散サーバ方式である Peh-GS を提案する．Peh-GS は，端末間通信を階層化した P2P ネットワークにより負荷分散と動的な同期を実現する．また，WebAssembly ベースのポータブル通信コアを用いることで，多様な環境での UGC 通信とゲーム状態の同期を可能にする．Peh-GS を WebRTC，WebSocket，WebAssembly を用いて実装し，実際のゲームエンジンを用いた統合検証を行った．

キーワード：MMORPG, UGC, P2P, WebRTC, WebAssembly

A UGC-friendly P2P-based Distributed Server Architecture in MMORPG

Zheng Chen[1]　Kenichi Kourai[1]

Abstract: Massively Multiplayer Online Role-Playing Games (MMORPGs) increasingly rely on user-generated content (UGC), which enables player-driven world building. However, traditional centralized servers suffer from load concentration and limited scalability. The growing volume of UGC has pushed these architectures to their limits. Therefore, distributed server architectures are required, but there are several issues to be addressed to support UGC-driven MMORPGs. This paper proposes Peh-GS, a distributed server architecture that leverages P2P communication to achieve high affinity with UGC in MMORPGs. Peh-GS employs a layered P2P network to enable load distribution and dynamic synchronization. In addition, it incorporates a WebAssembly-based portable communication core to support UGC communication and game-state synchronization across diverse environments. We have implemented Peh-GS using WebRTC, WebSocket, and WebAssembly, and confirmed its behavior with a real game engine.

Keywords: MMORPG, UGC, P2P, WebRTC, WebAssembly

1. はじめに

大規模多人数同時参加型オンライン RPG（MMORPG）は，数万人規模のプレイヤが同一世界でリアルタイムに交流・競争・協力するゲームのジャンルである．プレイヤが自ら世界を拡張し，遊び方を創出するユーザ生成コンテンツ（UGC）は，近年，ますます重要性を高めており，MMORPG の体験を多様化・深化させる主要な要素となっている．一方で，UGC は内容や発生位置がプレイヤの行動に応じて大きく変化し，局所的な集中や突発的な増加が生じやすいため，通信負荷や処理要求が偏在しやすい．そのため，UGC の体験はサーバアーキテクチャの性能と柔軟性に強く依存する．

[1] 九州工業大学

近年，GPU によるグラフィックス表現は飛躍的に向上した一方で，プレイヤ端末や周辺デバイスの計算資源は十分に活用されておらず，ゲーム世界の拡張性や自由度は依然として中央集権型サーバの処理能力に制約されている．従来の中央集権型サーバ方式では，UGC の偏在的かつ動的な負荷に対応することは難しく，UGC 主導型の大規模世界を維持する上でスケーラビリティやリアルタイム性の面で限界が生じる．そのため，分散型アーキテクチャの必要性が高まっており，P2P 方式を用いることで負荷分散や低遅延通信を実現することができる．しかし，UGC 主導型 MMORPG を支えるためには，従来方式では対処が困難な様々な課題を解決する必要がある．

本稿では，UGC の予測困難性・局所性・突発性に適応しつつ，MMORPG の世界状態を分散的に維持するためのサーバアーキテクチャである Peh-GS を提案する．Peh-GS はプレイヤの行動に基づいてゲーム空間を活動領域に動的に分割し，その上に階層的な P2P ネットワークを構築することで，負荷分散・局所同期・故障耐性を備えた世界状態管理を実現する．また，WebAssembly（Wasm）ベースのポータブル通信コアを用いることで，多様な端末の計算資源を利用可能にする．このように，Peh-GS は大規模・動的なゲーム世界の実現を支援することを目的とし，UGC の増大と多様化に伴って高まるプレイヤの自由度への要求に応えつつ，MMORPG のゲーム設計に新たな可能性を拓くことを目指す．

Peh-GS は，WebRTC を用いてセッションプレーンを構築し，階層型 Mesh トポロジと分散シグナリングにより Peer 間接続を管理する．また，WebSocket を用いてオーバレイプレーンを構築し，四分木に基づく分散ルーティングにより活動領域の探索と担当割り当てを行う．Wasm ベースの通信コアはブラウザおよびネイティブアプリケーションといった異なる環境から共通に利用することができる．ホストと Wasm 通信コアとの間では，Wasm の線形メモリと C ABI を用いて高速なデータ受け渡しを行う．加えて，非同期ランタイムを用いたイベント駆動処理をサポートする．Peh-GS を用いて，各機能の動作確認および，Godot Engine との統合検証を行い，正常に動作することが確認できた．

以下，2 章で UGC 主導型 MMORPG の背景と問題点を述べる．3 章で P2P を用いた分散アーキテクチャである Peh-GS を提案し，4 章でその実装を示す．5 章で Peh-GS を用いて行った実験について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. UGC 主導型 MMORPG

近年の MMORPG では，プレイヤが生成する UGC が世界構築の主要要素となっている．UGC はクエスト，建築物やスクリプトなど多様な形式を取り，開発者が事前に設計・配置する従来のゲーム資源とは異なり，内容・量・発生位置を事前に計画したり，統一的にモデル化したりすることが難しい．さらに，UGC の生成傾向はプレイヤ群ごとに大きく異なり，局所的に密集したり突発的に増加したりするため，通信負荷や処理要求が空間的・時間的に偏在しやすい．

従来の中央集権型サーバ方式では，このような偏在的な負荷に柔軟に対応することは難しい．MMORPG は数万人規模のプレイヤが同一ゲーム世界で同時に活動することを前提としており，状態が動的に更新される永続的な空間を維持するために，サーバが全プレイヤの状態を一元管理する必要がある．プレイヤや UGC の数が増加すると負荷が特定サーバに集中し，スケーラビリティや運用コストの面で限界が生じる．特に，UGC のように発生パターンが不規則で突発的なデータは，瞬間的な通信集中や負荷の偏在を引き起こしやすく，リアルタイム性や世界状態の一貫性を維持することが困難となる．

そのため，動的な負荷分散や局所的な処理の移譲が可能な分散型アーキテクチャの必要性が高まっている．P2P 方式はこのような分散型アーキテクチャの一つであり，Peer と呼ばれる端末同士が直接通信することで負荷分散や低遅延通信を実現することができる．近年では WebRTC などの技術により，ブラウザを含む多様なクライアント環境で P2P 通信が可能となり，ビデオチャット，ファイル共有，コンテンツ・デリバリ・ネットワーク（CDN），ゲームのルームマッチングなどに利用されている．しかし，現在の MMORPG における P2P の利用は小規模セッションに限定されており，大規模 MMORPG の広域で持続的な世界全体を扱うのは難しい．

例えば，多くのオンラインゲームではゲーム空間を静的なセルに分割し，各セルをサーバや処理ノードに割り当てる方式が採用されてきた．しかし，P2P においてこの方式を用いるにはいくつかの問題がある．まず，UGC の発生位置が動的に変化するため，負荷が特定セルに集中しやすい．また，プレイヤ密度の変動に追随できず，過負荷セルと過疎セルが発生する．さらに，セル境界が固定されているため，局所的な UGC の密集に対応できない．UGC 主導型の世界では，静的セル分割方式は適応性に欠け，動的な負荷分散が困難となる．

また，大規模な P2P ネットワークでは，ノードを接続する際に自身を登録し，他ノードを発見し，接続先を探索する仕組みが必要となる．しかし，これらの機能を中央集権的なサーバに依存すると，以下の問題が生じる．

- □ **初期接続要求の集中**：多数のノードが同時に接続・再接続を行うと，登録・発見要求が単一点に集中し，ボトルネックとなる．
- □ **単一障害点**：中央サーバが障害を起こすと，ノード発見や接続の形成が全面的に停止する．

- **スケール限界**：ノード数の増加に伴い，登録情報の管理コストや探索負荷が急増し，分散型アーキテクチャの利点が損なわれる．

MMORPG のように多数のプレイヤが動的に参加・離脱し，UGC の発生位置に応じて通信要求が変動する環境では，ノード登録・発見を中央集権的に扱う方式は適応性に乏しく，分散的に Peer を探索・接続できる仕組みが求められる．

以上を踏まえると，UGC 主導型 MMORPG を支えるためには，従来方式では対処が困難な以下の課題を解決する必要がある．

- **UGC の予測困難性・局所性への適応不足**
  UGC は発生位置や規模が予測困難であり，静的な空間分割や固定的な処理割当では負荷の偏在を解消できない．
- **大規模 P2P を組織するための構造の不足**
  フラットな P2P では Peer 数の増加に伴い管理負荷が増大し，動的に変化するプレイヤ密度に対応できない．
- **ノード登録・発見を中央集権的に扱うことによる接続集中**
  多数の Peer が動的に参加・離脱する環境では，登録・発見要求が単一点に集中しやすく，初期接続や再接続がボトルネックとなり，分散型アーキテクチャの利点が損なわれる．
- **UGC データの性質に応じた管理方式の欠如**
  更新頻度や重要度が異なる UGC データを一律に扱うと，通信量や保存コストが非効率となる．
- **多様な端末環境における通信処理の一貫性の欠如**
  ブラウザ・ネイティブ・モバイルなど実行環境が異なると通信処理を個別に実装する必要が生じ，挙動の不整合や保守負荷の増大を招く．

3. Peh-GS

本稿では，UGC の予測困難性・局所性・突発性に適応しつつ，MMORPG の世界状態を分散的に維持するためのアーキテクチャ Peer-enhanced hierarchical Game Synchronization（Peh-GS）を提案する．Peh-GS は，プレイヤの行動に基づく活動領域の動的分割を基盤とし，その上に階層的な P2P ネットワークを構築することで，負荷分散・局所同期・故障耐性を備えた世界状態管理を実現する．Peh-GS は，(1) プレイヤの行動に応じた活動領域の管理，(2) 活動領域を単位としたセッションプレーン，(3) 分散 Peer 発見を担うオーバレイプレーン，(4) UGC の更新特性に基づくデータ同期戦略，(5) WebAssembly ベースのポータブル通信コアの 5 つの要素から構成される．以下では，各要素について説明する．

3.1 活動領域

Peh-GS の基盤となる概念は，プレイヤの行動に基づいて形成される活動領域である．活動領域は，プレイヤ集団の位置・密度・移動傾向に応じて動的に生成・統合・分割される単位となる領域であり，UGC が局所的に発生するという特性に自然に追随する．従来の静的セル分割方式とは異なり，活動領域は以下の特徴を持つ．

- **動的分割**：プレイヤ間距離が拡大した場合に領域を分割し，局所的な負荷を分散する．
- **動的統合**：複数のプレイヤ集団が接近した場合に領域を統合し，相互作用を効率化する．
- **局所性の保持**：UGC が発生した領域に対して，その領域内の Peer が優先的に処理を担当する．

このように活動領域を動的に管理することで，UGC の発生位置に応じた負荷分散と，局所的な同期処理の効率化が可能となる．

3.2 セッションプレーン

セッションプレーンは，活動領域を単位として Peer を階層的に組織化する通信層である．Peh-GS では，Peer，Node（活動領域），Game（世界全体）の三層構造を形成し，世界状態の同期を分散的に行う．各活動領域には，以下の役割を持つ Peer が存在する．

- **Leader**：領域代表として外部領域との接続を担当し，領域間同期を行う．
- **Member**：領域内部の通信を担当し，UGC データやプレイヤ状態を同期する．

Leader はセッションプレーンを通じて選出され，領域間の接続構造を維持する．

セッションプレーンは，活動領域内外で異なる通信戦略を採用する．領域内部では，UGC の局所性に基づき，高頻度・低遅延の同期を行う．一方，領域間では，Leader を介した低頻度同期により，世界全体の整合性を維持する．これにより，広域な世界の整合性と局所的なリアルタイム性を両立しつつ，Peer 数の増加に伴う接続数の爆発を抑制できる．

3.3 オーバレイプレーン

オーバレイプレーンは，Peh-GS の基盤となる分散ルーティング層であり，Peer 登録・発見・接続先探索を中央集権的に扱う方式の問題を解消する．オーバレイプレーンは以下の機能を提供する．

- **接続要求の受け付け**：領域を中継する Router の役割を果たす Peer が存在し，他 Peer の接続要求を受け付けることで，中央集権的なシグナリングサービスを不要とし，初期接続の負荷を分散する．
- **分散ルーティング**：Router は位置情報に基づく分散ルーティングテーブルを保持し，近傍領域を効率的に

探索する．

- □ **Leader の発見**：セッションプレーンの Leader（領域代表 Peer）選出に必要な近傍領域情報を提供する．
- □ **故障耐性**：特定 Peer の離脱時に代替経路を探索し，接続構造を維持する．

オーバレイプレーンにより，Peer の参加・離脱が頻繁に発生する MMORPG 環境でも，スケーラブルで堅牢な Peer 発見と接続形成が可能となる．

3.4 データ同期戦略

Peh-GS では，MMORPG の世界状態を Hot，Warm，Cold の三種類に分類し，それぞれに異なる同期戦略を適用する．

- □ **Hot**：UGC データやプレイヤ状態などの近接同期データ．活動領域内の Peer 間で高頻度・低遅延の直接同期を行う．
- □ **Warm**：領域をまたいでも残る世界状態などの持続状態データ．活動領域の移動や再接続時に必要な範囲のみを差分同期する．
- □ **Cold**：静的データや生成規則などの基盤構造データ．初期ロードや低頻度の更新時のみ配布し，通常はキャッシュを利用する．

この階層化により，UGC を含む動的な世界状態に対して，局所的一貫性と世界の継続性を両立しつつ，通信量と保存コストを最適化できる．

3.5 ポータブル通信コア

Peh-GS は，多様な端末の計算資源を活用するために，通信処理に WebAssembly （Wasm）ベースのポータブル通信コアを用いる．Wasm を用いることにより，ブラウザとネイティブアプリケーションの双方で同一ロジックを実行することができる．Wasm によるオーバヘッドを削減するために，UGC データの高速処理とゼロコピー転送を行う．また，非同期イベントと内部処理の効率的な連携も可能にする．Wasm による統一的な通信基盤は，端末側での処理負荷の分散を容易にし，中央サーバへの依存を最小化する．

4. 実装

4.1 WebRTC を用いたセッションプレーンの構築

Peh-GS のセッションプレーンは，ブラウザおよびネイティブアプリケーションの双方で動作可能な WebRTC DataChannel を基盤として構築される．

4.1.1 通信トポロジ

WebRTC は本来，音声・映像ストリームの伝送を目的として設計されており，既存の通信トポロジは以下の三種類に大別される．

- □ **Mesh**：各 Peer が相互に直接接続する方式で，構成が単純である一方，参加者数に比例して接続数が増加し，Peer の負荷が急増する．
- □ **Multipoint Control Unit (MCU)**：中央サーバがメディアを合成して配信する方式で，端末負荷は小さいが，中央サーバがボトルネックとなる．
- □ **Selective Forwarding Unit (SFU)**：中央サーバがストリームを選択的に転送する方式で，MCU より軽量だが，依然として中央集権的な構造を持つ．

これらの方式は，オンライン会議やルーム制ゲームなど小規模セッションでは有効であるが，MMORPG のように多数の Peer が広域に分布し，UGC が局所的かつ動的に発生する環境には適していない．Mesh では接続数が過剰に増加し，MCU や SFU では中央集権的すぎるため，いずれもスケーラビリティと負荷分散の観点で限界がある．

そこで Peh-GS では, Mesh を拡張した階層型 Mesh トポロジを採用し，活動領域を単位とした階層構造を WebRTC 上に構築する．これにより，各 Peer が維持すべき接続数を抑えつつ，領域内部では高頻度同期，領域間では Leader を介した低頻度同期を実現する．図 1 のように，従来の Mesh では Peer 間の接続数が $O(N)$ に増加するのに対し，階層型 Mesh では活動領域ごとに Peer がグループ化され，Leader を中心とした $O(1)$〜$O(\log N)$ 程度の接続数に抑制できる．これにより，UGC の局所性に応じた効率的な同期が可能となる．

4.1.2 シグナリング

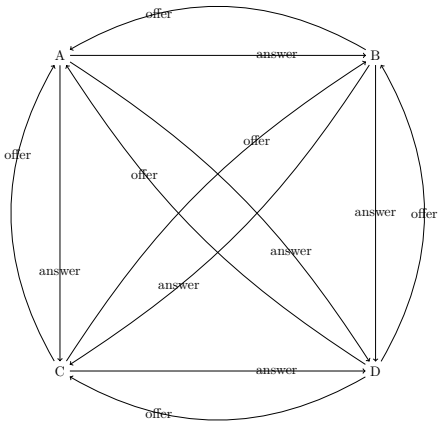
WebRTC による Peer 間接続には，Session Description Protocol（SDP）と Interactive Connectivity Establishment（ICE）を用いたシグナリングが必要となる．

- □ **SDP**：通信方式・チャネル情報など，会話の内容を記述する．
- □ **ICE**：NAT 越えのための候補経路を列挙し，到達可能な経路を探索する．

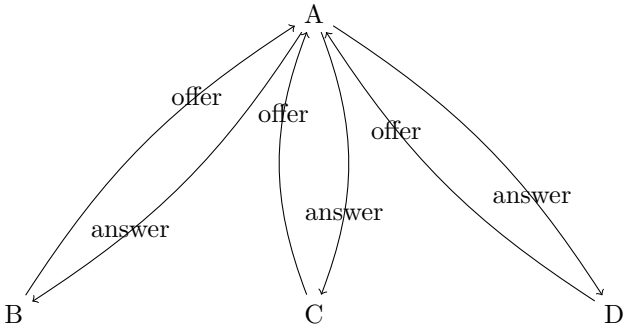
通常，SDP/ICE はシグナリングサーバを介して交換されるが，MMORPG のように多数の Peer が同時に接続・再接続を行う環境では，シグナリングサーバへの負荷集中が問題となる．

そこで Peh-GS では，シグナリング処理に Webhook 機構を導入し，サーバ負荷を大幅に軽減する方式を採用する．Webhook を用いたシグナリング処理は図 2 に示すように行われる．

(1) Peer A は SDP/ICE を直接送信せず，Webhook にアップロードする．
(2) Peer A は Webhook のダウンロードリンクのみをシグナリングサービスへ送信する．
(3) シグナリングサービスサービスはリンクを Peer B に転送する．
(4) Peer B は Webhook から SDP/ICE を取得し，接続を確立する．



(a) 従来の Mesh



(b) 階層型 Mesh

図 1 Peh-GS における階層型 Mesh トポロジ

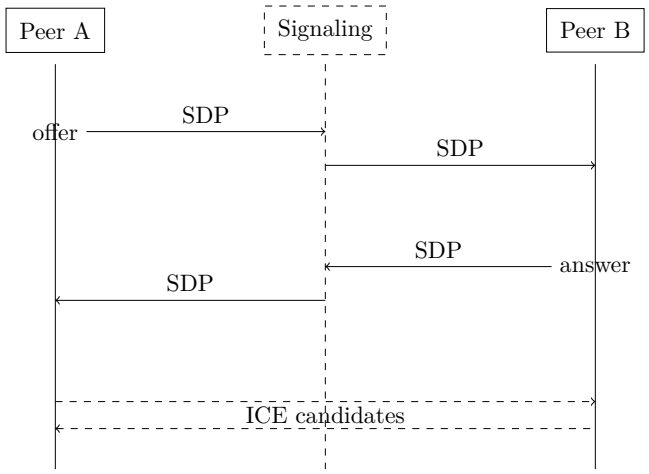


図 2 Webhook を用いたシグナリング処理

この方式により，シグナリングサービスはリンク転送のみを担当し，SDP/ICE の実体データは外部 CDN，WebDAV，あるいは Discord チャンネルなどの Webhook に委譲される．これにより，シグナリングの帯域負荷と同時接続数を大幅に削減でき，Peer 間接続のスケーラビリティが向上する．

4.2 WebSocket を用いたオーバレイプレーンの構築

Peh-GS のオーバレイプレーンは，分散 Peer 発見と接続先探索を担う基盤層であり，WebSocket を用いて構築される．Distributed Hash Table（DHT） [1] および Balanced Tree Overlay Network（BATON） [2] の設計思想を参考に，Peh-GS では四分木に基づく分散ルーティング構造を採用する．各ノードは，自身が担当する四分木区画を示すタグ T[ABCD]+ を保持する．T はルート区画，A〜D は四分木の象限を表し，タグは階層的に拡張されることでノード位置に応じた一意の識別子となる．

新しく参加するノードは，自身の座標 (x, y) に基づき，四分木上で担当すべき区画が決まるまで再帰的に探索を行う．手順は以下の通りである．

(1) ステップ 1：トップレベル T への接続
ノードはまずルート区画 T に接続し，(x, y) が属する象限（A〜D）を判定する．対応する区画が未担当であれば，その区画を管理し、タグ T + 象限文字（例：TA）を取得する．

(2) ステップ 2：担当ノードへの接続と再帰的探索
対象象限（例：TA）を管理するノードに接続し，下位区画の中から (x, y) が属する象限を再判定する．未担当であれば新規ノードがその区画（例：TAB）を管理し，担当済みであればさらに下位へ進む．

(3) ステップ 3：未担当区画が見つかるまで細分化
担当済み区画が続く場合，担当ノードに接続しながら下位区画を再帰的に探索し，未担当の象限が見つかった時点でその区画を管理する．
（例：T → TB → TBC → TBCA）

(4) ステップ 4（任意）：親区画担当者の再評価
兄弟区画のノードの負荷や遅延を比較し，より健全なノードを上位区画の担当者に昇格させることで，トポロジ全体の負荷分散を図ることも可能である．

4.3 Wasm ベースのポータブル通信コア

Peh-GS の通信処理は Rust により実装し，Wasm バイナリとしてコンパイルする．これにより，同一の通信コアをブラウザおよびネイティブアプリケーションといった異なるホスト環境から共通に利用できるポータブル通信コアとして構成する．WebRTC を用いるセッションプレーンと WebSocket を用いるオーバレイプレーンのいずれにおいても，通信の中核となるロジックはこの Wasm 通信コア内に実装されており，ホスト側は環境ごとの差異を吸収する薄いラッパのみを持てばよい．

4.3.1 線形メモリと C ABI を用いた通信コアの利用

Wasm はホストと共有する線形メモリを持ち，Wasm 通信コア内で確保したバッファをホストが直接参照できる．Peh-GS では，この特性を利用してデータなどの転送におけるコピー回数を削減し，ブラウザとネイティブアプリケーションの双方で同一のデータレイアウトを扱えるようにする．そのために，通信コア側でバッファを確保し，

ポインタと長さを整数値としてホストへ返す．ホストは線形メモリ上の該当領域を直接読み書きする．不要になったバッファは通信コア側で明示的に解放する．

Wasm 通信コアの関数は extern ”C” 形式の ABI を用いて呼び出すことができ，ポインタ・長さ・識別子などのプリミティブ値のみを引数として受け取る．これにより，JavaScript が動作するブラウザと Rust，Go，C++ などで実装されたネイティブアプリケーションの双方から，同一の Wasm 通信コアを呼び出すことができる．

4.3.2 非同期ランタイムとイベント駆動 Future

ブラウザのようなシングルスレッド環境では，Wasm 通信コア内でブロッキング処理を実行するとレンダリングが停止する可能性がある．そのため，Peh-GS では Rust の Future とホスト側のイベントループを統合し，非同期処理をイベント駆動でスケジュールする．

(1) Wasm 通信コアは Future を生成するが，自身でブロッキング処理を実行せず，処理要求をイベントとしてホストへ委譲する．
(2) ホストはブラウザであれば Promise，ネイティブであれば非同期タスクとして処理を継続する．
(3) 処理完了時にホストが Wasm 通信コアの線形メモリに結果を書き込み，事前に登録された外部関数を呼び出して完了を通知する．
(4) Wasm 通信コアは通知を受けて該当タスクの Future をポーリングし，再開する．

このようなイベント駆動型の Future の実装により，WebRTC の接続確立やデータチャネル通信，オーバレイプレーン上のメッセージ処理などを，ホスト環境のイベントループと協調しながら非同期に実行できる．

4.3.3 Wasm 通信コアの読み込み

Peh-GS はゲームエンジンとして Godot Engine [3] に対応している．Godot のネイティブエクスポートでは，Rust で書かれた GDExtension ライブラリを介して Wasm 通信コアを読み込む．具体的には，Wasm ランタイムとして Wasmtime [4] を組み込み，Wasm バイナリをロードしてインスタンス化し，Godot 側と双方向に通信できる実行環境を構築する．

一方，Godot のブラウザエクスポートでは GDExtension 自体が Wasm ターゲットをサポートしているものの，別途 Wasm ランタイムを組み込む必要がある．そこで Peh-GS では，ブラウザが備える WebAssembly Web API [5] を直接利用する方式を採用する．

具体的には、Peh-GS は Godot の ExportPlugin を拡張し，Wasm バイナリをエクスポートリソースとして Web ビルドに埋め込む．ブラウザ実行時には，次の手順で Wasm 通信コアを読み込む．

(1) Godot のファイルシステム API を用いて埋め込まれた Wasm バイナリを読み出し，その内容をバイト列として取得する．
(2) JavaScriptBridge を介して JavaScript 側にバイト列を渡し，ArrayBuffer を構築して Wasm モジュールの入力とする．
(3) 別の JavaScript のスクリプトにより，WebAssembly Web API を用いて ArrayBuffer から Wasm インスタンスを生成し，Godot 側と双方向に通信できる環境を構築する．

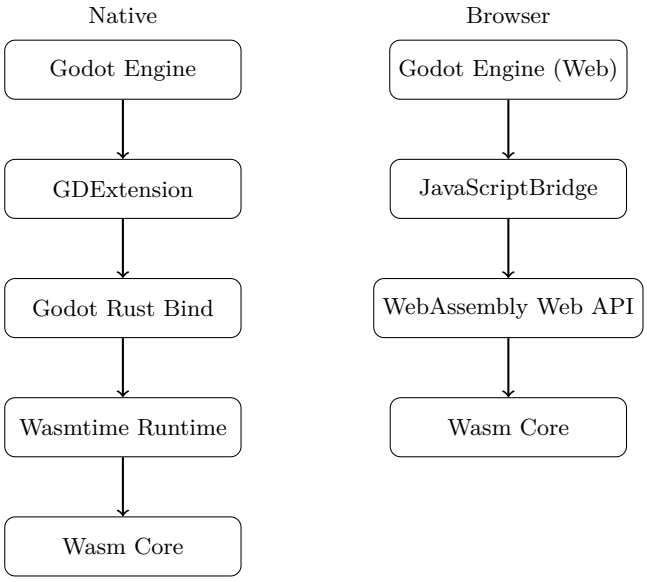


図 3 Wasm 通信コアの読み込み経路

以上のネイティブ環境とブラウザ環境における Wasm 通信コアの読み込み経路を，図 3 に示す．

この仕組みにより，ネイティブアプリケーション版と同一の Wasm 通信コアをブラウザ環境でも利用でき，セッションプレーンおよびオーバレイプレーンの通信処理を統一的に実行できる．

5. 実験

Peh-GS の Wasm ベースの通信コアがブラウザおよびネイティブ環境で共通に動作し，セッションプレーンおよびオーバレイプレーンの通信処理を一元的に実行できることを確認する実験を行った．また，実際のゲームエンジンの Godot Engine 上で各構成要素が相互に連携することを確認する実験を行った．本実験には，表 1 に示すホスト環境を用いた．シグナリングに用いる Webhook には Discord チャネルを用いた．

5.1 Wasm 通信コア

Rust で実装した通信コアを wasm32-unknown-unknown ターゲットとしてビルドし，ブラウザおよびネイティブアプリケーションから共通に利用できることを確認した．それぞれの環境で，線形メモリを介したバッファ共有を行い，ホスト側が追加コピーなしにデータを参照することができ

表 1　実験環境

	ノード
CPU	AMD Ryzen 7 5800H
メモリ	32 GB
ネットワーク	Wi-Fi 6
OS	Windows 11 25H2
Rust	1.94.1
Wasm	Wasmtime 32.0.1
Godot	v4.6.2.stable.steam
ブラウザ	Microsoft Edge 147.0
TypeScript	6.0.2

た．また，extern ”C” ABI による関数呼び出しを正常に行うことができ，ポインタ・長さ・識別子の受け渡しが正しく処理された．非同期ランタイムにおいて，Wasm 内の Future がホストのイベントループと連携し，ブロッキングを発生させずに処理を継続することができた．これらの結果から，Wasm 通信コアが Peh-GS の基盤として十分に機能することを確認した．

5.2　セッションプレーン

セッションプレーンでは，Wasm 通信コアを介して WebRTC DataChannel を確立し，Peer 間で UGC データおよびプレイヤ状態を交換できることを確認した．新規 Peer を P2P ネットワークに接続する際には，WebHook を用いたシグナリングにより SDP/ICE 情報が正しく交換された． その後，DataChannel を確立して，双方向通信を安定して行うことができた．また，Leader と Member の役割分担に基づき，領域内外の通信を正しくルーティングすることができた．

5.3　オーバレイプレーン

オーバレイプレーンでは，四分木に基づく分散ルーティングが正しく構築され，Peer の参加・離脱に応じてタグが一意に割り当てられることを確認した．具体的には，新規 Peer が座標に基づいて適切な領域を担当し，タグ T[ABCD]+ が正しく生成された．その際に，既存 Peer が担当する領域に対して再帰的に接続し，下位領域が正しく探索された．一方，Peer の切断時には代替経路が探索され，ルーティングが維持された．

5.4　Godot Engine との統合検証

Wasm 通信コアを統合した GDExtension を用いて，Godot Engine のゲームループ内で通信処理が正しく動作することを確認した．まず，Godot のブラウザエクスポートにおいて，Wasm 通信コアが正しく読み込まれ，ブラウザ上で正常に動作することが確認できた．次に，Godot から Wasm 通信コアの関数を呼び出し，Wasm の線形メモリ上のデータを直接参照することができた．また，WebRTC と WebSocket のイベントが Wasm 通信コア経由で Godot へ正しく伝達され，Godot 側でゲームロジックを更新することができた．Wasm 通信コアでの非同期処理が Godot のシングルスレッド・レンダリングループを阻害しないことも確認できた．

5.5　性能

Wasm 通信コアとホスト間のデータ転送性能，および Godot Engine とブラウザ間のデータ転送性能について測定した．

5.5.1　Wasm–Host 間データ転送性能

Wasm とホスト間で 1 MB のランダム文字列を転送し，線形メモリを介したデータ共有方式とパイプ型転送方式の処理時間を比較した．その結果，データ共有方式では 42.13 ms，パイプ型転送方式では 42.49 ms となり，安定して転送できることを確認した．

なお，両方式の違いとして，データ共有方式は処理結果を格納するために Wasm 側で新たにメモリ領域を割り当て，そのアドレスをホストへ返却するのに対し，パイプ型転送方式は呼び出し元が渡したバッファ領域をそのまま再利用し，メモリ容量の面では利点がある。

5.5.2　Godot–Web 間データ転送性能

Godot Engine とブラウザ間で 1 MB のランダム文字列を 100 回転送し，ArrayBuffer を用いたバイナリ転送方式と，テキスト列シリアライズ方式の性能を比較した．その結果，ArrayBuffer を用いた方式では 81.06 ms，テキスト列送信は 372.88 ms となった．

ArrayBuffer を用いた方式はテキスト列送信に比べて約 4.6 倍高速であり，これらの結果から，ブラウザ環境においても Wasm 通信コアがバイナリデータを効率的に処理でき，ネイティブ環境と同様に高い転送性能を維持できることが分かった．すなわち，Peh-GS の通信基盤はブラウザとネイティブの双方で一貫した性能を発揮し，クロスプラットフォームの実行環境として十分に機能することが確認できた．

5.5.3　新規 Peer の登録遅延

新規 Peer がゲーム空間に参加し，領域情報およびタグが割り当てられるまでの処理時間を測定した．その結果，ネイティブ環境では 60 ms 程度，ブラウザでは 93 ms 程度となった．ブラウザ環境では若干の遅延が見られるものの，いずれも 100 ms 未満であり，リアルタイム性を損なわずに Peer の参加処理が行えることを確認した．

6.　関連研究

MMORPG における空間管理方式として，MOPAR [6] やその改良版 [7] に代表される，セルベースの Area of Interest（AOI）管理方式が提案されている．これらの研究では，ゲーム空間をセルに基づいて管理し，セルごとに

Master を配置する階層型 P2P アーキテクチャを採用している．Fujita による改良版では，プレイヤ密度に応じたセルの分裂・統合を導入し，セル単位での負荷分散を可能にしている．

これらの方式は階層構造を用いた分散管理という点で Peh-GS と似ているが，空間の分割単位が地理的セルに依存している点が異なる．Peh-GS では，地理的セルではなくプレイヤグループの活動範囲を単位とする動的な活動領域を採用している．さらに，セル構造とは独立した分散四分木ルーティング層を導入することで，空間構造と通信構造を分離している．

P2P ネットワークにおける代表的な分散ルーティング方式として，Chord [1], Pastry [8], Kademlia [9] などの DHT が提案されている．また，BATON [2] は木構造に基づく階層的なルーティングを実現している．これらの方式は大規模分散環境における探索効率に優れるが，MMORPG のようにプレイヤの分布や相互作用が動的に変化する環境を直接扱うものではない．Peh-GS では，四分木構造を Peer の登録・発見のための軽量なシグナリング層として利用し，ゲームロジックとは独立した分散ルーティングを構築している．

WebRTC の通信トポロジとしては，Mesh，MCU，SFU が一般的である．Mesh は小規模セッションに適するが，参加者数に比例して接続数が増加するため大規模環境には不向きである．MCU や SFU は中央集権的構造を持ち，サーバ負荷がボトルネックとなる．これらの方式はいずれも，広域かつ持続的な世界を扱う MMORPG の要件を満たさない．Peh-GS の階層型 Mesh は，WebRTC を用いながらも接続数を抑制し，大規模環境に適応するための新たな構造を提供する．

WebAssembly はブラウザおよびネイティブ環境で動作可能なポータブル実行形式として注目されている．線形メモリを用いたゼロコピー通信や，C ABI によるホスト連携も利用されている．しかし，WebRTC と WebSocket の双方を単一の Wasm 通信コアとして実装したものは我々の知る限り，存在しない．Peh-GS は，Rust で実装した通信処理を Wasm 通信コアに集約し，ブラウザとネイティブアプリケーションの双方で共通に利用できる通信基盤として構築している．

7. おわりに

本稿では，UGC 主導型 MMORPG における動的・局所的な相互作用を効率的に扱うため，活動領域を基盤とした階層型 P2P アーキテクチャ Peh-GS を提案した．Peh-GS は，セッションプレーンによる階層的な Peer 間接続，および，オーバレイプレーンによる分散四分木ルーティングを提供する．また，WebRTC，WebSocket，Wasm を統合したポータブル通信コアを提供し，ブラウザとネイティブアプリケーションの双方で共通通信基盤として用いることができる．Godot Engine との統合検証を通して，Peh-GS が実際のゲームエンジンで動作可能であることを示した．

今後の課題として，現時点では試作的な実装と初期的な検証に留まっている活動領域の動的管理機構や，UGC の更新頻度・重要度に応じたデータ同期戦略の高度化が挙げられる．また，本稿では 1 台のノードだけを用いて基本的な動作検証を行ったが，大規模環境における性能評価や，プレイヤ密度が急変する状況でのスケーラビリティ検証を行う必要がある．さらに，分散型アーキテクチャ特有の安全性やチート対策についても，通信コアおよびプロトコル設計の観点から検討を行う．

参考文献

[1] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *Proceedings of SIGCOMM 2001*, pp. 149–160 (2001).
[2] Jagadish, H. V., Ooi, B. C., Tan, K.-L., Vu, Q. H. and Zhang, R.: BATON: A Balanced Tree Overlay Network for Peer-to-Peer Networks, *Proceedings of VLDB 2005*, pp. 661–672 (2005).
[3] Godot Engine Team: Godot Engine - Free and open source 2D and 3D game engine, Godot Engine Team (online), available from ⟨https://godotengine.org/⟩ (accessed 2026-04-10).
[4] Bytecode Alliance: Wasmtime, Bytecode Alliance (online), available from ⟨https://wasmtime.dev/⟩ (accessed 2026-04-10).
[5] W3C: WebAssembly JavaScript Interface, W3C (online), available from ⟨https://webassembly.github.io/spec/js-api/⟩ (accessed 2026-04-10).
[6] Yu, A. P. and Vuong, S. T.: MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games, *Proceedings of NOSSDAV 2005*, pp. 99–104 (2005).
[7] Fujita, S.: Load Balancing of Peer-to-Peer MMORPG Systems with Hierarchical Area-of-Interest Management, *International Journal of Networked and Distributed Computing*, Vol. 3, No. 3, p. 177 (2013).
[8] Rowstron, A. and Druschel, P.: Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems, *Proceedings of Middleware 2001*, pp. 329–350 (2001).
[9] Maymounkov, P. and Mazieres, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, *Proceedings of IPTPS 2002*, pp. 53–65 (2002).