P2P環境でのネットワークゲーム向け負荷分散機構の提案

公原 勝彦 安本 慶一 伊藤 実
奈良先端科学技術大学院大学情報科学研究科

あらまし 本稿では，ピアツーピア (P2P) の環境で特定のサーバを設置することなく多人数参加型ネットワークゲームを実現することを目的として，ゲームで発生するイベントの登録・通知機構を，幾つかのゲーム参加者の端末に割り当て，分散処理させる機構を提案する．従来，ゲーム空間を均等に分割し，分割されてできた複数の領域を，固定数のサーバに対応させる方式や，P2P 環境で幾つかのプレイヤの端末に割り当て処理する方法が提案されている．しかし，これら従来の方法を P2P 環境にそのまま適用した場合，ある領域のプレイヤ数およびそこで発生するイベントの数が多くなると，その領域を担当するノードに過度の負荷を強いることになり適当でない．また，P2P 環境では，各ノードがゲームから離脱したときの対処法についても考慮する必要がある．本稿では，均等分割した各領域のプレイヤ数とそこで発生するイベントの発生頻度の積を監視し，それがある閾値を越えた時には，領域内のプレイヤの集合を部分集合に動的に分割し，新たに割り当てたノードに分散処理させることにより，各ノードの負荷を一定水準以内に抑える方式を提案する．また，イベント通知の配送のためのオーバレイネットワークの構築と中間ノードが離脱した時の回復機構について述べる．

A Load Distribution Mechanism for Multi-player Games on P2P Networks

Katsuhiko Kouhara Keiichi Yasumoto Minoru Ito
Graduate School of Information Science, Nara Institute of Science and Technology

Abstract In this paper, we propose a load distribution mechanism for multi-player network games on peer-to-peer networks, where game events are delivered to game players through players' nodes (terminals) in a distributed manner without specific servers. There are some studies on load distribution mechanisms for network games where a virtual space is divided to multiple sub-areas so that several servers can manage those areas in parallel. However, if we apply those exiting techniques to network games on P2P networks, some peer nodes may be overloaded when the numbers of events and players increase in an area. Moreover, in P2P networks, some server nodes may leave suddenly. In the proposed method, we let each server node periodically monitor the current load level as the product of the number of players and the number of events occurred. When the load level exceeds a threshold, the set of players is recursively divided to multiple sub-sets so that the load level for managing each sub-set is less than a threshold. Some new nodes are dynamically assigned to those sub-sets to deliver game events to players in each subset. We have developed an algorithm to construct an overlay network to deliver game events where each delivery path is dynamically recovered even when a node suddenly leaves.

1 はじめに

近年のインターネット接続回線の広帯域化ならびにPCおよび高性能ゲーム端末の普及により，数万人規模のユーザが同じ空間を共有し，互いにインタラクションを行うネットワークゲームが人気を集めている．

現在のネットワークゲームは，主としてクライアント・サーバによる集中制御をもとに構築されていることが多い．サーバでの集中制御はゲーム状態の管理のしやすさ，セキュリティの確保のしやすさなどの利点がある一方で，サーバに負荷が集中するという欠点を持つ．この欠点を補うためにゲーム空間を可視領域 (あるユーザにとって視覚的に見える範囲) に分割し，分割した領域を複数のサーバで管理することにより負荷を分散する方法が提案されている [1, 2, 4]．文献 [1] では，ゲーム空間の X, Y, Z 座標，ゲームイベントの種類などの各属性値について固定長の範囲に分割し，連続する範囲を論理リング上に並べたノードに分散管理させる（すなわち，イベントの発生のたびに，そのイベントがリング上を担当ノードに到達するまで転送され，担当ノードがその範囲でのイベントの通知を希望するプレイヤに通知を行う）方法を提案している．しかし，一般に，ゲーム空間におけるプレイヤの分布は片寄っており，プレイヤ密度の高い領域は時間とともに変化すると思われるため，この方式では，ある領域で発生するイベントの数が多くなると，その領域を担当するサーバの負荷が大きくなる．一方，文献 [4] では，3 次元仮想空間を可視領域に分割し，個定数のサーバによりこれらの領域で発生するイベント情報を分散管理し，負荷が増大したサーバでは，可視領域を動的に狭めることにより対処する方法が提案されている．しかし，P2P 環境でネットワークゲームを実現する場合，固定的なサーバは使えず，プレイヤ端末にサーバの肩代わりをさせる必要があるため，端末にか

かる負荷をある水準以下に抑えれることが望ましい．また，P2P ネットワーク上では，プレイヤがゲームをやめオーバレイネットワークを離脱することによる情報伝達パスの切断をどう回復するかも問題となる．

上記の問題に対し，本稿では，均等分割した各領域を担当するノードの負荷がある水準を越えた時には，領域内のプレイヤへのイベント通知処理を新たに割り当てた複数ノードで分散処理させることにより，各ノードの負荷を一定水準以内に抑える方式を提案する．

提案手法では，文献 [1] に基づき，プレイヤの中から必要数を可視領域担当ノードとして選出し，イベント登録・通知用に，論理リングに基づいたオーバレイネットワークを構築する．各可視領域担当ノードは，ある周期で領域内のプレイヤ数とイベント発生数の積 (イベント通知メッセージの配送回数に相当) をモニタし，その値が指定された閾値を越えたら，領域内のプレイヤを幾つかの集合に分割するとともに，分割後のプレイヤの集合の担当ノードを選出し割り当てる．なお，分割された各集合に対しても，これらの処理を再帰的に適用することで，一つのノードが担当するイベント通知メッセージの配送回数を一定値以内に制限できる．元のノードと新たに割り当てたノードとの間の接続関係はツリー型のオーバレイネットワークとして構築する．あるプレイヤがイベントを発生させると，イベント通知メッセージが可視領域担当ノードに配送され (初回のみ，担当ノードに到達するまでリング上を配送される)，ツリーをたどって，メッセージがコピーされ，最終的に，ツリーの葉ノードから領域内の全てのプレイヤにイベント通知メッセージが配送される．

ノードの離脱によるイベント通知配送経路の切断を回避するため，提案方式では，論理リング上の各領域の担当ノードに対しバックアップノードを用意し，それらの間で定期的に情報を交換させることで，バックアップノードへのすばやい切り替えを行う．また，ツリー上のノードが離脱した場合には，新たな代替ノードを確保してそれに切り替えるまでの間，親ノードに代行処理を行わせることで対処する．

提案手法により，一般的な P2P 環境で，可視領域間でプレイヤの数やイベント発生頻度の差が大きい場合でも，各ノードの負荷を一定値以下に小さくできることを，シミュレーションにより確認した．

2 基本方針

本稿では，ネットワークで接続された複数のプレイヤ間で同一の仮想空間と時間を共有する，鳥瞰型のロールプレイングゲーム (RPG) を想定する．仮想空間は，背景と複数の オブジェクトからなるものと定義する．ここで，オブジェクトは，プレイヤや他の移動体および静止物であり，それぞれ，属性を持つ．オブジェクトの属性を変化させる出来事をイベントと呼ぶ．イベントの例として，ユーザの移動や，他のオブジェクトへの働きかけ，オブジェクトの色・形状の変化などがある．

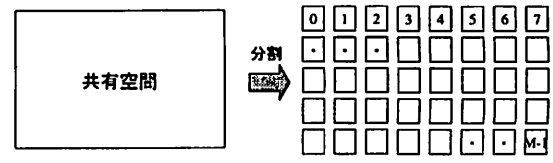

図 1: 仮想空間の分割

上記で定義されるようなゲームにおいて，複数プレイヤによるゲーム進行の一貫性を保証するには，以下を満たす必要がある．

- 仮想空間のある同じ範囲（仮想空間全体またはその部分領域）にいる全てのプレイヤは，常に同じゲーム状態（領域内に存在する全てのオブジェクトの位置とその属性）を保持する．
- その範囲で発生したイベントは，範囲内にいるプレイヤにリアルタイムで伝達される．
- 連続して発生した一連のイベントについては，その発生順序が，同一範囲内の全てのプレイヤで一意に保たれる．

本稿では，特別なサーバを用いないピアツーピア環境において上記を実現するため，以下の方針を採用する．

- 発生したイベントの影響範囲をプレイヤから見える範囲（ 可視領域と呼ぶ）に限定する．
- 図 1 に示すように，仮想空間を均等な可視領域に分割し，それぞれの領域でのイベントの登録・通知を，ある基準で選ばれた幾つかのプレイヤ端末に分散して担当させる．
- イベント通知機構として，publish/subscribe モデル [7] を用いる．すなわち，ある領域の担当ノードは，担当領域でのイベント発生 (publish) 時に，イベント通知登録 (subscribe) しているプレイヤにのみイベントを通知する．
- ゲームにおける時間の流れにタイムスロットによる同期機構を設けることでプレイヤ間でのイベントの発生順序を保存する．

3 ネットワークゲーム向け負荷分散機構

本稿では，プレイヤのリストを保持するロビーサーバ S の存在を仮定する．また，共有仮想空間は M 個の可視領域に分割されているとする（図 1）．

3.1 ゲームへの参加とリングの構築

ゲームへの参加者（プレイヤ）は，まず，S に接続し，自分の端末の ID，IP アドレス，利用可能なネットワーク帯域，処理能力を登録する．それに対し，S は，以下の処理を行う．

(1) S は，ゲームの参加者数 n が $n < \alpha \cdot M (\alpha \geq 1)$ の間は，ゲームを開始せずユーザの参加を募る．

(2) ゲームの参加者数が $n \geq \alpha \cdot M$ になったら，S は，M 個の可視領域でのイベント通知を担当する M 個のノード（以後，担当ノードと呼ぶ）を選ぶ．その際，ノード間の遅延時間が小さく，かつ，処理能力，ネットワーク帯域が大きいものを選択する．

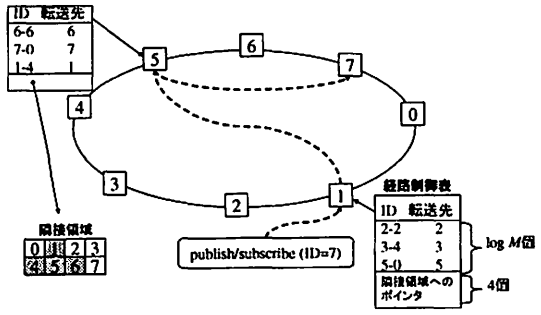

図 2: 経路制御表を用いたメッセージの転送

(3) 選択した M 個のノードのそれぞれに，図 1 の可視領域 $0, ..., M-1$ を対応させ，図 2 のような論理リングを構築する．リング構築の際には，隣接領域の担当ノード間の遅延が与えられた閾値 D 以内であるよう割り当てを行う．リング上の各ノードには，Chord [6] で用いられているサイズ $\log M$ の経路制御表を持たせる（図 2）．従って，各プレイヤからのイベント通知登録 (subscribe) およびイベント発生通知 (publish) は，図 2 に示すように，$O(\log M)$ 回のホップで担当ノードに到達できる．経路制御表には，仮想空間における上下左右の隣接領域の担当ノードへのエントリを含ませる．これにより，プレイヤが可視領域を越え移動した時でも，送信したメッセージが，移動先の可視領域の担当ノードに常に 2 ホップで到達できる．

(4) M 個のノードが選択されたら，S はゲームの開始を許可する．S は，以後参加を希望するプレイヤに対し，リング上のノード（以後，エントリノードと呼ぶ）を一つ選び通知する．

3.2 イベント情報の配送

提案方式では，publish/subscribe モデルを用いて，各プレイヤが可視領域（と隣接領域）で発生したイベントだけを受信することにより通信量の削減を図る．

3.2.1 publish/subscribe メッセージの内容

各プレイヤはゲーム参加時に通知されたエントリノードに subscribe メッセージを送信する．subscribe メッセージには，(1) プレイヤの ID，(2) IP アドレス，(3) イベント情報を通知してほしい領域の ID のリスト，(4) メッセージの有効時間，が含まれる．subscribe メッセージは，リング上を担当ノードに到達するまで転送され，担当ノードにおいて保持される．

各プレイヤは何か行動を行うたびに，その内容（イベント）を他のプレイヤに通知するための publish メッセージを発行する．publish メッセージには，(1) 自分の ID，(2) 対象オブジェクト（プレイヤ）の ID，(2) イベントの種類，(3) 共有空間上の座標，が含まれる．publish メッセージは，リング上の担当ノードに到達すると，そこでの全ての subscriber に配送される．

3.3 可視領域担当ノードの動的負荷分散

各可視領域での負荷（subscribe しているプレイヤ数×単位時間あたりに発生するイベント数）が予め指定した閾値 C を越えた場合には，リング上の該当ノードを根とする木構造のネットワーク (以後，負荷分散木と呼ぶ) を構築することにより負荷分散を実現する．リング上のノードに subscribe しているプレイヤ (subscriber) を分割し，木の複数のノード (ロビーサーバにより選択されたノード) に分散して担当させることにより，単位時間あたりに処理すべきイベントの数を削減する．

3.3.1 負荷分散木の構築

リング上の各ノードは，あらかじめロビーサーバから，子ノードの候補となるプレイヤノードの情報を受けとっていると仮定する．リング上のノード A は，それらのノードに対し，“ping” などにより遅延時間が最小となるノードを 2 つ選び，子ノードとして接続する．

ノード A は，以下の手順で負荷分散木を構築し，子ノードへ subscriber の割り当てを行う．

(1) ノード A は，現在の subscriber の集合を，ほぼ同数になるよう半分に分割し，分割された 2 つの subscriber の集合を，A に接続された 2 つの子ノードに送信する．この際，3.6 節で述べる回復機構のため，ノード A には，どの子ノードがどの subscriber を担当しているかの情報（subscriber リスト）を保持させる．

(2) ノード A に publish メッセージが来たら，両方の子ノードに転送する．また，A に subscribe メッセージが来たら，子ノードのうち，subscriber が少ない方に転送する．

(3) 負荷分散木のある葉ノードの負荷が高くなったら (subscriber の数×単位時間あたりに発生するイベント数が閾値 C を越えたら)，手順 (1) に従い，subscriber の集合を分割し，子ノードに割り当てることで再帰的に負荷を分散する．

(4) 負荷分散木のある中間ノードを根とする部分木の subscriber の数×単位時間あたりのイベント数が閾値 C 以下になったら，子ノードの統合を行う．そのため，各中間ノードは，イベントの発生頻度，子ノードの担当する subscriber の集合をモニタさせる．

(5) 負荷分散木の各中間ノードには，3.6 節で述べる回復機構のため，孫ノードの IP アドレスおよび subscriber 集合を保持させる．そのため，各中間ノードは親ノードに対し，subscriber 集合，自ノードの IP アドレス，子ノードの IP アドレスを含むメッセージを定期的に送信する．

3.4 リングと負荷分散木を用いたメッセージの配送

プレイヤがゲームに参加して最初に publish(あるいは subscribe) メッセージを送信するとき，および，共有空間上の不連続な移動（ワープ）を行ったときなどプレイヤが自分の現在座標の担当ノードを知らない場合，自分の知っているリング上のノード（ロビーサーバから指示されたエントリノードあるいは，前にいた座標の担当ノード）に publish(subscribe) メッセージを送信する．この場合，図 3 の点線で示すようにメッセージがプレイヤに配送される．この際，リング上の

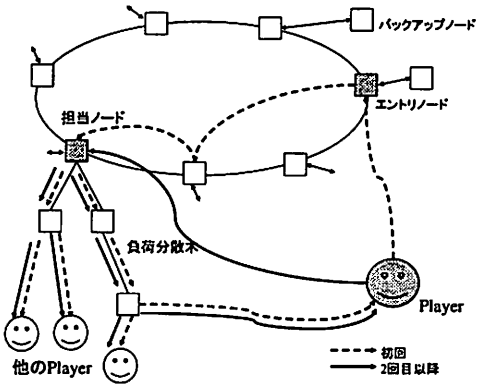

図 3: publish メッセージの配送

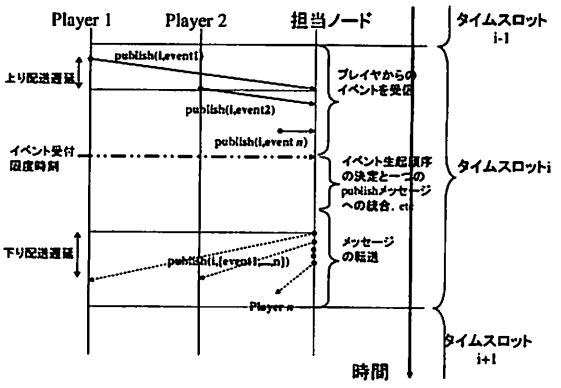

図 4: timeslot

担当ノードは，以下に述べる理由により，publish メッセージを受け取ると，自分の IP アドレスをメッセージに付加し子ノードへ転送する．

一度他のプレイヤから publish メッセージを受信すると，そのメッセージには，所属する可視領域の担当ノードの IP アドレスが含まれているため，以降はそのノードに publish(あるいは subscribe) メッセージを送信する．この場合，図 3 の実線で示すように，メッセージはリング上のノードを通らず，リング上の担当ノードから負荷分散木の各ノードを通してプレイヤに配送されるため，配送遅延を短縮できる．

また，プレイヤが共有空間上を連続的に移動し可視領域の境界を越えた場合，新たな可視領域に subscribe する必要があるが，3.1 節で述べたように，リング上の各担当ノードは経路制御表に，共有空間の隣接領域の担当ノードのエントリを保持しているため，subscribe メッセージを 2 ホップで適切なノードへと転送できる．

3.5 タイムスロットを用いた同期機構

ネットワークゲームではゲーム空間および時間の流れの一貫性が要求される．ここでいう一貫性とは，(1) イベント生起時刻の同時性：あるイベントが発生すると，近隣の全てのプレイヤが，同じ時刻同じ場所でそのイベントが発生したことを知覚すること，(2) イベント生起順序の一貫性：全てのプレイヤにとって，一連のイベントの生起順序が同一であること，を指す．すなわち，ネットワークの遅延等により，あるプレイヤのあるオブジェクトへの操作（例えばドアを開ける）が一部のプレイヤに伝わらなかったり，それにより，他のプレイヤが矛盾した操作（開いたドアをさらに開けるなど）を実行するようなことがあってはならない．

上記 (1)，(2) を実現するため，提案手法では，図 4 のように，ネットワークゲームにおける時間の流れを固定長のタイムスロットに分割し管理する手法を採用する．ここで，全てのプレイヤノードは，NTP などを用いることにより，ある程度の正確さで各タイムスロットの始まりを認識できると仮定する．各タイムスロットで起こすことのできるイベントは各プレイヤにつき高々一つまでとし，各ノードは一連のタイムスロットを順序番号（タイムスロット番号）により識別するものとする．一般に，ゲームで遊ぶユーザは自分の入力が反映されるまでに 400ms 以上経過すると「結果が反映されていない」と感じることが知られている [3]．そのため，本稿では 200ms 程度のタイムスロットを想定する．

各タイムスロットでは，各プレイヤはタイムスロット番号を付加した publish メッセージを発行する．各領域の担当ノードは，領域内のプレイヤからの publish メッセージを受け取り，3.2 節で述べた方法で，subscribe している全てのプレイヤに配送する (図 4)．この際，各担当ノードおよびプレイヤノードは，メッセージに付加されているタイムスロット番号と現在のタイムスロット番号を比較し，一致する場合のみそのメッセージを受理し，そうでない場合メッセージを廃棄する．以上により，上記 (1) が実現される．

連続するイベントが，異なるタイムスロットで発生した場合は，全てのプレイヤでイベントの生起順序は同じになる．一方，同じタイムスロットにおいて複数のプレイヤによる複数のイベントが発生した場合，順序を一意に決定する必要がある．ここでは，担当ノードが受け取った publish メッセージを，図 4 のイベント受付限度時刻（タイムスロットの終了時刻から，下り配送遅延の最大値とメッセージ処理時間の和を引いた時刻）まで蓄積しておき，与えられた基準によりイベントの生起順序を決定することとした．順序の決定方法として，例えば，publish メッセージが発行された時間（タイムスタンプなどにより実現可能）順にソートする，あるいはランダムに順序を決定するなどが考えられる．また，順序付けされた一連の publish メッセージから順序付のイベントリストを構成し一つの publish メッセージに集約する．集約した publish メッセージを，subscribe しているプレイヤに配送することで，メッセージ転送のオーバヘッドを削減することができる．この際，イベント受付限度時刻より後に受け取られた publish メッセージは破棄される（イベントを発行したプレイヤは同じタイムスロットで受け取った publish メッセージから，イベントが破棄されたことを知るので，必要なら次のタイムスロットでもう一度イベントの発行を試みることができる）．集約された publish メッセージは，担当ノードから，負荷分散

木を辿り，各葉ノードからプレイヤに配送される．

また，配送経路上で publish メッセージが失われ，あるプレイヤノードで受信できなかった場合，同じタイムスロット内，あるいは次のタイムスロットに，紛失したメッセージの再送を要求する．この場合，最大1タイムスロット分の遅れが生じるが，イベントの生起順序は保たれるため，ゲームの一貫性を保持できる．上記を実現するため，担当ノードには，数タイムスロット分の publish メッセージを保持させる．

3.6 障害発生時の回復機構

ネットワークゲームでは，プレイヤの新規参加，ゲームからの離脱が頻繁に起こることが想定される．ここでは共有空間のある領域を担当しているノードが突然離脱する場合の対処法について，リング上のノード，負荷分散木上のノードの場合に分けて説明する．

リング上のノードが離脱した場合

リング上の M 個のノードに，それぞれ自分のバックアップとして1つのノードを持たせることで対処する (図 3)．なお，リング上のノードとそのバックアップノードは同時に停止しないと仮定する．ここで，バックアップノードは，リング上のノードが保持している経路制御表，担当している領域に subscribe しているプレイヤの情報 (subscribe 情報)，負荷分散木の子ノードの情報を定期的にバックアップする．リング上のノードは，これらの情報に変化があった時に，バックアップノードに変更情報を通知する．

バックアップノードは対応するリング上のノードを監視し，一定時間 (5 タイムスロットなど) 応答がない場合は，ノードが停止したものと断定し，自信がリング上のノードになるとともに，ロビーサーバに問い合わせて，バックアップノードを割り当てる．また，リング上の他のノードの経路制御表を書き換えるための「経路変更メッセージ」をリング上に巡回させる．なお，経路回復時間の短縮のため，隣接領域の担当ノードには直接経路変更メッセージを送信する．リング上の他のノードは経路変更メッセージを受け取ると，経路制御表の該当エントリを書き換える．

負荷分散木上のノードが離脱した場合

負荷分散木上の各中間ノードは，子ノードが停止していないかどうかを定期的に監視しているものとする．負荷分散木上のあるノード A が停止した場合を考える．

A が葉ノードの場合，A の親ノードは A の停止を検知すると，A の subscriber への publish メッセージの配送を一時的に代行する (3.3 節で述べたように，各中間ノードは子ノードの subscriber リストを保持している)．後は，3.3 節で述べた方法で，A の代わりのノードを新規に割り当てる．

A が中間ノードの場合，A の親ノード B はそれを検知すると，一時的に publish メッセージを A の子ノード (B の孫ノード) に直接転送する (3.3 節で述べたように，各中間ノードは，子ノードおよび孫ノードの IP アドレスを知っている)．その後，B はロビーサーバに問い合わせて，新たに1ノードを割り当て，A の代わりとする．

上で述べた方法では，数タイムスロットの間プレイヤにイベント情報が伝わらず，その間ゲームの進行が止まってしまう．各ノードがゲームをやめてオーバレイネットワークから離脱する際に，バックアップノード (負荷分散木の場合は親ノード) に通知するようにすることで，新しいノードへの切り替えを，シームレスに行うことができる．

4 実験と評価

4.1 負荷分散の効果

提案手法では，プレイヤノードが可視領域でのイベント通知の配送を行うため，そこでの負荷が問題になる．そこで，可視領域内のプレイヤ数 n，および，可視領域を分散処理するノードの数 m を変化させることで，負荷がどう変わるかを実験により調べた．

実験には，A, B, C の 3 台の PC (A, C は，高岳製作所 MiNT PC, FreeBSD4.2R, PentiumIII 800MHz, 512MB RAM, B は VineLinux2.0, Athlon XP 2000+, 256MB RAM) を用いた．A には，n 台のプレイヤが 250ms 毎に担当ノード B に向けメッセージ (各メッセージ 50 バイトとした) を送信することを模倣するプログラムを，B には，各プレイヤから送信されたメッセージを n 人のプレイヤ (C) に転送するプログラムを[1]，C には，B から送信されたメッセージを単に受信するプログラムを，それぞれ実行した．A, B, C は 100BASE-TX で接続されており，配送遅延は約 0.4ms である．A-B 間,B-C 間の通信には TCP を用いた．

n を 80〜200，m を 1〜8 まで変化させた時の，B の負荷 (top コマンドによる CPU 占有率およびメッセージ転送処理時間) の推移を図 5, 6 に示す[2]．図 5, 6 から，負荷分散を行わない場合 (m=1 の場合)，プレイヤ数が 120 を越えると，CPU 占有率が 8%以上，処理時間が 50ms 以上と高くなり，P2P ネットワークのあるノードが処理を負担するのは難しいことが分かる．一方，$n = 200$ の場合でも，$m = 8$ で負荷分散すると，CPU 占有率，処理時間ともに 6%，35ms 未満となりネットワークゲームでは 400ms 以上の遅延が発生するとゲームプレイに影響が出る [3] ことを考慮すると実用的な範囲に収まっていることが分かる．

また n が 120 の場合において m を 2, 4, 8 と変化させた場合に，メッセージ複製のオーバーヘッドによりゲームの処理性能がどの程度上下するかを調べた．実験にはベンチマークプログラムとして FINAL FANTASY XI Official Benchmark2 ver1.1 を使用し，

[1] ここでは，メッセージの集約は行わず，送られて来たメッセージを単純に n 人のプレイヤにコピーし送っている．従って，負荷分散を行わない場合 ($m = 1$)，B が転送するメッセージは n^2 個となる．

[2] 図 5 で m=1 のグラフにおいて，n>120 の範囲が示されていないのはこの条件では実験に用いた PC が負荷に耐えられずダウンしたためである．

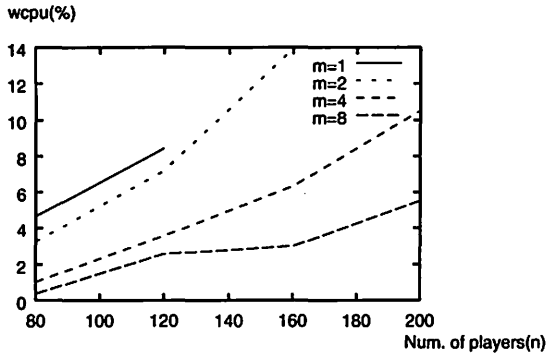

図 5: CPU 占有率の推移

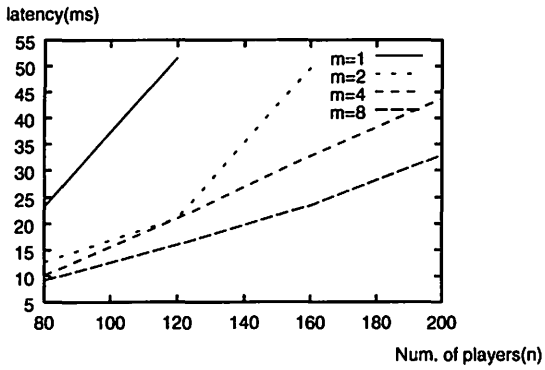

図 6: メッセージ処理時間の推移

上記と同じ構成で実験を行った（ただし，ノードBには，VAIO PCG Z1R/P (Pentium M 1.5GHz 512MB RAM) を使用し，このマシンでベンチマークプログラムを並列実行した）．なお，このベンチマークのスコアが1000以上であれば問題なくゲームがプレイできるとされている．実験の結果，通信を行なわない場合のスコア1549に対し，通信を行う場合，m=2の時1461，m=4の時1506，m=8の時1518というスコアが得られた．本ベンチマークプログラムは3D処理を多様しているため，一概には判断できないが，提案手法により，ゲームプレイにほとんど影響がない程度に負荷分散できると思われる．

4.2 遅延時間

3.5節で述べたように，プレイヤがイベントを発行してから，同じ可視領域の他のプレイヤがそれを受け取るまでの配送遅延は，M を可視領域の数，m を可視領域の分散数，α をオーバレイリンクの伝送遅延，β を中間ノードでの処理遅延とすると，初回で，$\alpha \cdot (\log M + \log m + 1) + \beta$，2回目以降で，$\alpha \cdot (\log m + 1) + \beta$ と表すことができる．ここで，m=8，β=50msとすると，2回目以降のメッセージの配送遅延は，$4\alpha + 50$ となる．一方，オーバレイネットワークのノード間の遅延時間は，国内で15ms程度，アジア内で50ms程度，日米間で150ms程度であることが分かっている．この場合，国内およびアジア内では，配送遅延を250ms程度に抑えることができることが分かる．また，可視領域の担当ノード間，負荷分散木のノード間のオーバレイリンク遅延が短くなるようノードを選ぶことで，メッセージ配送遅延をさらに短縮できるため，$m > 8$ の場合を扱うことも可能である．

以上より，提案方式を用いることで，P2P環境で実用上十分な性能を持つ実時間ネットワークゲームを実現できると考えられる．

5 まとめ

本稿では，P2P環境でのネットワークゲームにおいて，各プレイヤの端末にゲームイベントの通知を担当させる場合の，負荷分散方式について提案を行った．提案方式では，共有空間を分割した各領域について，プレイヤ数およびイベント発生頻度を監視し，各担当ノードの負荷が高くなる場合には，担当領域に複数の新たなノードを動的に割り当てることにより，各ノードの負荷を与えられた水準以内に制御できる．また，担当ノードがオーバレイネットワークから離脱する場合のバックアップ方式と，プレイヤ間でゲームイベントの発生順序の一貫性を保つためのタイムスロットに基づいた方式について提案した．

予備実験により，提案方式によって，各ノードの負荷が一定値以内に抑えられること，一般的なネットワークにおけるメッセージ配送遅延を考慮した場合に，タイムスロットの長さを通常のリアルタイムRPGで使用可能な範囲に抑えることができることなどを確認した．今後，より詳細なシミュレーションの実施，ミドルウェアおよびその上で動作するゲームのプロトタイプの作成を行い，提案方式の有用性を評価したい．

参考文献

[1] Ashwin R. Bharambe, Sanjay Rao and Srinivasan Seshan: "Mercury: A Scalable Publish-Subscribe System for Internet Games", *Proc. of 1st Workshop on Network and System Support for Games (NetGames2002)* (2002).

[2] W. Broll: "Distributed Virtual Reality for Everyone Framework for Networked VR on the Internet", *Proc. of 1997 IEEE Virtual Reality Annual Intl. Symposium (VRAIS'97)*, pp. 121 - 128 (1997).

[3] T. Henderson: "Latency and Behaviour on a Multiplayer Game Server", *Proc. of 3rd Intl. Workshop on Networked Group Communication (NGC2001), LNCS2233*, pp. 1–13 (2001).

[4] 井芹，堀，藤川，下條，宮原：多人数参加型ネットワークアプリケーションの広域ネットワーク環境における利用実験，信学技法，IN2000-121，pp. 21–28 (2000).

[5] A. Rowstron, A.-M. Kermarrec, M. Castro and P. Druschel: "SCRIBE: The design of a large-scale event notification infrastructure", *Proc. of 3rd Intl. Workshop on Networked Group Communication (NGC2001), LNCS2233* (2001).

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan: "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", *Proc. of ACM SIGCOMM'01*, pp. 149–160 (2001).

[7] A.S. Tanenbaum and M. v. Steen: "Distributed Systems, Principles and Paradigms", Prentice-Hall (2002).