F-045

分散オンラインゲームにおける同期方式の研究

An Efficient Synchronization Mechanism for Distributed Online Games

小川 健太郎† ユリウス オネシムス‡ 小林 洋†
Kentaro Ogawa Yulius Onesimus Hiromi Kobayashi

1． まえがき

ネットワークを通じて離れたプレイヤとの協力，対戦を行うゲームでは，プレイヤ間での通信に遅れが発生してしまう．遅れが大きい場合には，受け取ったパケットをすぐに処理してしまうと，プレイヤ間で違った処理が行われる可能性がある．例えば，プレイヤ 1 がプレイヤ 2 に向かって攻撃をし，プレイヤ 2 はその攻撃を避けた時，プレイヤ 2 から 1 へのパケットの遅れによりプレイヤ 1 側の処理ではプレイヤ 2 に当たり，プレイヤ 2 側の処理では当たらないというようなことが起こりえる．このような問題の解決のためプレイヤ間では同期が取られる．ところが，これについて定量的データを評価した文献が，特にサーバを介さずにプレイヤ同士で通信を行う Peer to Peer 型ゲームの同期方式についてはほとんど見当たらない．本研究では同期方式として提案されている Trailing State Synchronization（TSS）と呼ばれる方式で，特に，プレイヤ間でやり取りするメッセージがコマンドの場合（コマンド方式）と，キャラクタの位置の場合（位置方式）の違いにより，処理時間等の変化についての評価を行った．

2． Trailing State Synchronization

本研究で採用している TSS[1]という同期方式では，プレイヤからの入力命令をすぐに処理し画面に表示するのではなく，少しだけ処理を行う時間を遅らせて，その遅れの間に通信を行うことでプレイヤ間での同期を取る方式である．

この方式ではどれだけ処理を遅らせるかを決める複数個の同期遅延（Synchronization delay）と，各同期遅延の時間分，ゲームの状態を保存する Trailing State（以下 State）というデータリストを使用する．同期遅延はそれぞれ別の値に設定し，State に保存する時間が短い順に State0，State1，…と呼ぶことにする．特に State0 は，Leading State とも呼ばれ，この State に保存されている最も古いゲーム状態で，画面表示に使用する．図 1 に Trailing State を 2 つ，それぞれの同期遅延を 50ms と 100ms にした場合の例を示す．

同期の手順は，以下の 4 つのステップで行われる．

(1) プレイヤの入力が行われた時間（タイムスタンプ）と通信に使用するパラメータを，他の全てのプレイヤに送信する．このとき入力を行ったプレイヤ自身も，未だ入力をゲームへは反映させない．

(2) 他のプレイヤのメッセージを受け取ったプレイヤはそのデータを全てのデータリストに保存する．もし State の同期遅延を過ぎていた場合には，その State への保存はしない．

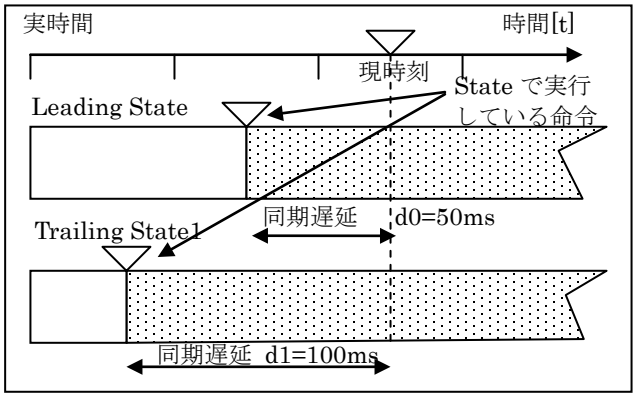

図 1 TSS の概要

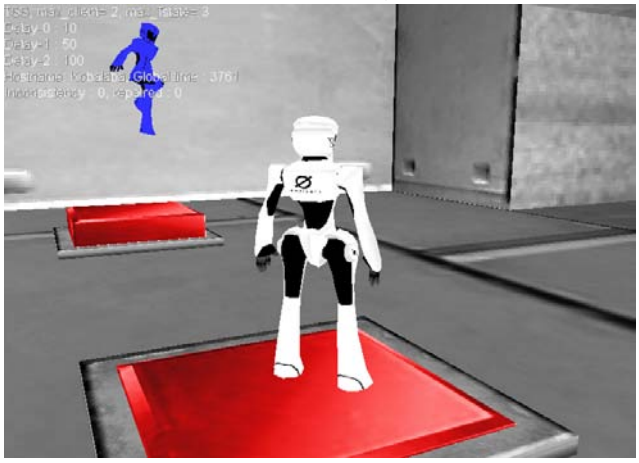

図 2 ゲームのプレイ画面

(3) Leading State に保存されたデータから，[現在の時間]－[Leading State の同期遅延] に一致するタイムスタンプのデータをゲームへ反映させる．

(4) State に保存されたデータから，[現在の時間]－[State の同期遅延] に一致するタイムスタンプのデータと，一つ前の State の同じタイムスタンプのデータを比較し，違いがあれば修正ステップへ移行する．

State N と State N+1 間で修正の必要なデータが発見された場合，修正は以下のステップで行われる．なお，時間をロールバックする部分のアイデアは，タイムワープ機構[2]と同じである．

(1) 修正の必要なデータを発見した場合，State N を使用しそのデータのタイムスタンプの状態までゲームをロールバックする．

(2) State N＋1 から State N に正しいデータをコピーする．

(3) State N の修正の行われたデータから，State N-1 の同期遅延までのデータで，ゲーム状態の再計算を行う．

(4) N を 1 ずつ減らしながら，Leading State まで 1~3 を繰り返し行う．

†東海大学 大学院 工学研究科 Tokai University
‡株式会社 スパイク

3. 作成したアクションゲーム

評価を行うために作成したゲームは，複数人のプレイヤが自キャラクタを操作し，協力してゴールを目指すというものである．ゲームの画面を図 2 に示す．送信メッセージの種類は，コマンド方式か，位置方式かのどちらかが選択でき，これにタイムスタンプが付けられ送信される．画面描画は Leading State のデータを使用し行われる．そして，同期のずれに起因するデータ修正を必要に応じて行う，という一連の手順を繰り返し行ってゲームは進行する．

4．評価

今回作成したゲームで，データ修正の発生率と，State 数や同期遅延の大きさ，送信するメッセージの違いなどにより処理時間にどのような変化が起きるかを測定した．まずデータ修正の発生率の測定では，2 つの State を作成し，Leading State の同期遅延を 0,1,2,…,10,20,40，State1 の同期遅延は常に 100 として，通常処理の行われた回数とデータ修正の発生回数を測定し,データ修正の発生率を算出した．図 3 はコマンド方式での結果で，同期遅延を 7 以上に設定した場合，データ修正がほとんど発生していないことがわかる．これは通信により発生する遅延がある時間以上になることがほとんどないためと考えられる．なお，これについては，位置方式でも同様と考えられる．

次に State 数の変化による処理時間の変化の測定では，State を 2～11 個作成し，各同期遅延を 10×2^{n}（n=0～10）として正常な処理が行われたときの処理時間と，データ修正が行われたときの処理時間を測定した．正常処理時の結果は図 4 に示すようにコマンド方式，位置方式のどちらも State 数を増やすと処理時間が増加している．またそれぞれの方式を比べた場合，コマンド方式は位置方式よりも処理時間が早い．これはコマンド方式の場合，正常な処理が行われたときの処理は，他のプレイヤへのメッセージの送信と，Leading State からデータを取り出し処理をするだけで済むのに対して，位置方式の場合は，それらの処理に加えて，プレイヤの入力からキャラクタの位置を計算する処理があるためと考えられる．

データ修正時の処理時間の早さはコマンド方式では正常処理時から大幅に処理時間が増えたが，位置方式ではほとんど変わっていない．これは位置方式の場合，修正処理は State に保存された位置データを使用し，ゲーム状態の再計算を行うのみであるのに対して，コマンド方式ではそれに加えてキャラクタの位置を修正開始時にロールバック後，保存されたコマンドデータから，キャラクタの位置の再計算が必要になり,大幅に処理時間が増加したと考えられる．

また同期遅延に対する位置方式のデータ修正時の処理時間を図 5 に示す．測定を行う前は，データ修正時の処理時間は同期遅延の大きさに比例すると考えていたが，一定以上の同期遅延を設定した場合，図 1 からわかるように修正の必要なデータが少なくなっていくため，State 数の増加に合わせて,データ修正には一部の State しか使われなくなり，処理時間の増加が緩やかになったと考えられる．

5．おわりに

State 数や同期遅延の値は，大きくする程同期に効果的であると考えやすいが，測定結果から，同期処理には余り効果がない場合があり，また，処理時間の低下を招くため，プレイヤ間での通信の状況に合わせて設定する必要があることがわかった．そのため今後の課題としては，ゲームの実行時に発生する通信遅延に合わせて，プログラムの実行中に動的に State を作成し，不要になれば破棄するようにアルゴリズムを改善することが考えられる．またその場合，遅延の発生にどのように対応するかの検討が必要である．これらの課題については，現在対応中である．

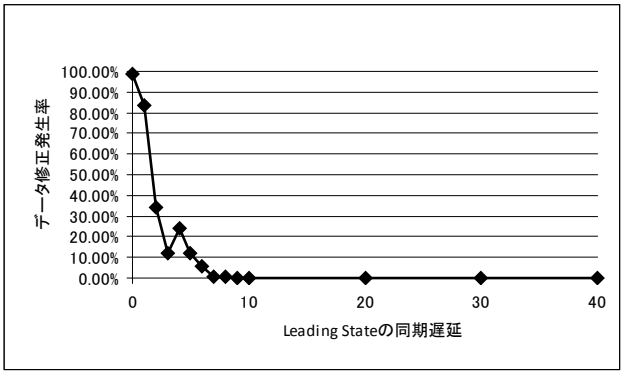

図 3　データ修正の発生率

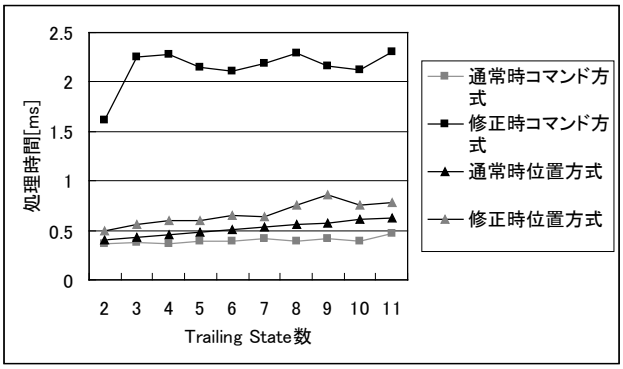

図 4　Trailing State 数に対する処理時間

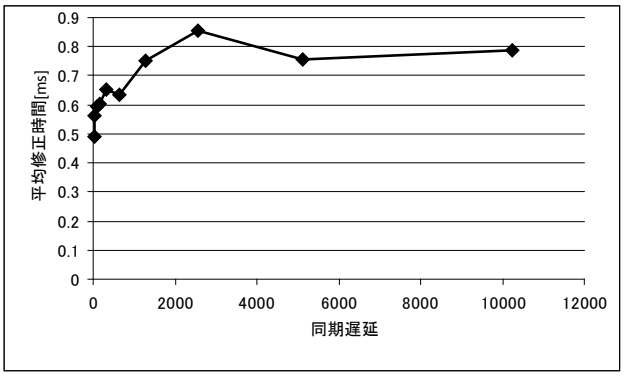

図 5　位置方式での同期遅延データ修正時の処理時間

参考文献

[1] E. Cronin, A. R. Kurc, B. Filstrup, S. Jamin, “An Efficient Synchronization Mechanism for Mirrored Game Architectures”, NetGames2002, pp.67-73（2002）.

[2] D. Jefferson, “Virtual Time”, ACM Trans. Programming Languages Syst., Vol. 7, No.3, pp.404-425 (1985) .