

シミュレーションキャッシングフレームワークの実装

山本 優^{1,a)} 西村 祐介¹ 福間 慎治¹ 森 眞一郎^{1,b)}

受付日 2015年6月30日, 採録日 2015年12月7日

概要: 我々は, 遠隔地の高性能コンピュータ上でのシミュレーションにおいて対話的な操作を可能とする遠隔インタラクティブシミュレーションシステムの実現を目指して研究を行っている. このような遠隔インタラクティブシミュレーションでは, ネットワーク通信による遅延が発生し操作の妨げとなる. この問題の改善手法として我々はシミュレーションキャッシングの研究を行ってきた. 本論文では, シミュレーションキャッシングシステムのフレームワーク化を行い, 様々なシミュレーションシステムに応用できるシステムの構築を行った.

キーワード: インタラクティブシミュレーション, 遠隔シミュレーション, 遅延隠蔽, 一貫性制御

Implementation of Simulation Caching Framework

YU YAMAMOTO^{1,a)} YUSUKE NISHIMURA¹ SHINJI FUKUMA¹ SHIN-ICHIRO MORI^{1,b)}

Received: June 30, 2015, Accepted: December 7, 2015

Abstract: In order to realize human-in-the-loop scientific computing in cloud like environment, we have to conquer the problem of network latency. For this purpose, we propose a simulation model which we referred to it as “Simulation Caching”. The infrastructure for Simulation Caching is a sort of cooperative cloud where a high performance remote server cooperates with a moderate scale local server to provide an immediate and reasonable response to the operator’s interactive steering of the simulation. To hide the latency to the remote server, Simulation Caching lets the local server caches a part of the simulation from the remote server and performs the duplicated simulation concurrently with the remote server, while keeping the accuracy of the cached simulation. This paper reports about the prototype implementation of Simulation Caching Framework and its use case examples with some performance report.

Keywords: interactive simulation, remote simulation, latency hiding, consistency control

1. はじめに

近年の計算機性能の急速な向上にともない, インタラクティブな実時間シミュレーションへの期待が高まっている. フライトシミュレーション [1] や航空管制シミュレーションのようにコンピュータ上のシミュレーション結果を操作者が直接体感し, その反応として対話的にシミュレーションをステアリング可能なシミュレーションの形態は “human-in-the-loop simulation” あるいは “interactive simulation” と呼ばれる. 近年さかんに研究が行われてい

る, 災害時の緊急避難シミュレーションなどもその一例である. 従来, このようなシミュレータ上で行われてきたシミュレーションは主に離散事象シミュレーションであったが, これをスーパーコンピュータ上の科学技術計算のシミュレーションにも拡張する試みも進められている [2], [3], [4]. しかし, 実時間での対話的なステアリングまでを考慮した研究は始まったばかりである [5], [6]. これに対して我々は, 遠隔地のシミュレーションサーバで行うシミュレーションの一部を通信遅延の少ない操作端末の近くで行い, 2つのシミュレーション結果を連携させて利用し通信遅延を隠蔽する “シミュレーションキャッシング” [7] の研究を行ってきた.

このようなシミュレーションキャッシング技術を応用した対話型遠隔シミュレーションが容易に実行可能となれ

¹ 福井大学大学院工学研究科
Graduate School of Engineering, University of Fukui, Fukui
910-8507, Japan

a) yyu@sylph.fuis.u-fukui.ac.jp

b) moris@u-fukui.ac.jp

ば、スーパーコンピュータなどの高性能計算資源活用 の敷居が下がり、実時間指向の幅広いユーザ層への HPC の普及と発展が期待できる。そこで本研究では、これまで遠隔流体シミュレーション用に開発したシミュレーションキャッシングの技術に機能拡張を行い、様々なシミュレーションに適用可能な汎用性の高いシミュレーションキャッシングフレームワークの実装を行った。さらに本論文では、実際にフレームワークを用いてシミュレーションキャッシングを他のシミュレーションへ適用させ、評価を行った結果の報告も行う。

2. 関連研究

従来より、遠隔地にある均質/非均質な計算機資源を連携した協調計算のフレームワークに関する研究が行われている。たとえば、原子力研究所では異機種計算機間で MPI をつかった協調計算を可能にする通信ライブラリ Stampi [8] が開発されており、アーキテクチャの異なるスーパーコンピュータを連携したマルチフィジックスシミュレーションなどの研究が行われた。また最近では、大学などが所有するプライベートクラウドシステムを連携させ広域分散型のインタークラウドシステムを実現するための遠隔連携技術に関する研究 [9], [10] も行われている。しかしながら、これらの従来研究はスループット重視のアプリケーションを指向したものが大半であり、我々がターゲットとしている実時間指向のアプリケーションへの適用に関する議論は十分に行われていない。

実行中の大規模シミュレーションの遠隔モニタリングに関する関連研究としては、SIMONsystem [11], RSVLIB [12], VisTrace [13] などがあるが、いずれもシミュレーション自体のステアリングに関する機能は有していない。

統合的なシミュレーション・可視化環境の関連研究としては、複数のシミュレーションを統合し、シミュレーション実行、可視化、対話的な分析とステアリングをサポートする環境として World Lines [14] がある。これは境界条件の異なる複数のシミュレーションを 1 つのプラットフォーム内で制御し、過去のステアリング情報を確認・再計算が可能である。しかしながら、大規模計算機との遠隔協調シミュレーションへの応用については検討が行われていない。

スーパーコンピュータベースの遠隔シミュレーションステアリング環境の関連研究例としては COVISE [15] がある。これはシミュレーションの実行、可視化機能を統合するための分散ソフトウェア環境であり、機械設計などにおける工程（設計、モデリング、メッシュ、シミュレーション、可視化、解析）が 1 つのプラットフォームで実行可能となる。また、Web ベースのインタフェースを導入することでユビキタスなアクセスと協調作業を可能にしている。しかしながら、実時間でのシミュレーションステアリングについては考慮されていない。

これらの先行研究に対して、大規模計算機上で実行中のシミュレーションに対し、遅延隠蔽技術を用いて遠隔地から実時間でのステアリングと可視化を可能にするシミュレーションフレームワークの提案を行うところに本研究の特徴がある。

3. 研究背景

3.1 対話型遠隔シミュレーション

手元にある計算機では実行不可能な大規模シミュレーションを、遠隔地にある高性能な計算サーバ上で実行し、ネットワークを通じて計算条件や結果を対話的に通信することで遠隔操作するシステムである。前提として、リモートサーバは対象とするシミュレーションの実行に対して十分な計算性能を持っているものとする。このシステムは、クライアント側を操作端末としてシミュレーションをステアリングし、サーバ上でステアリングされたシミュレーションの新しい条件から数値計算を行い、結果データを操作端末にフィードバックすることによって操作端末の負荷を軽減し、手元にある計算機だけでは不可能な大規模シミュレーションを実現するものである。

3.2 シミュレーションキャッシング

シミュレーションキャッシングで想定する対話型遠隔シミュレーションは、遠隔地の高性能シミュレーションサーバ（リモートサーバ）と、ユーザの操作や結果の提示を行う操作端末と、その近傍にあるパーソナルシミュレーションサーバ（ローカルサーバ）で構成されるクライアントサーバ型のシステム [7] である（図 1）。通信遅延が存在する環境下においても、一定間隔でのデータ出力を補償する。

3.2.1 遅延隠蔽手法

シミュレーションキャッシングではまず、シミュレーションをパイプライン化しスループットを向上をさせる。しかし、計算条件が入力されてからその計算条件が反映されたシミュレーション結果を得るまでの遅延時間は依然と

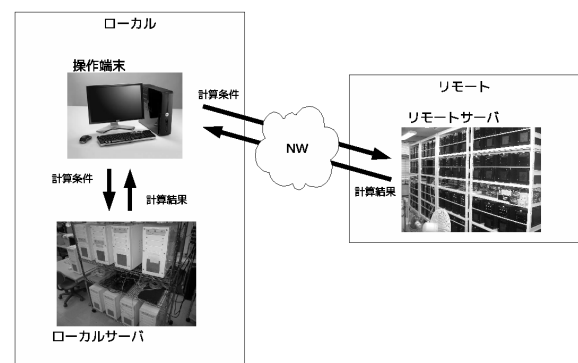


図 1 シミュレーションキャッシングを用いた対話型遠隔シミュレーションシステム

Fig. 1 Interactive remote simulation system using simulation caching.

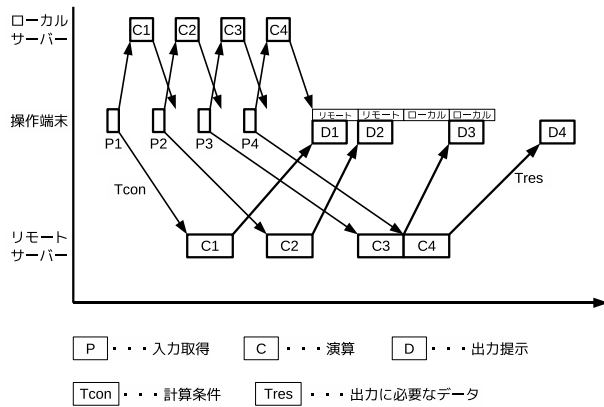


図 2 シミュレーションキャッシングの動作

Fig. 2 Behavior of simulation caching.

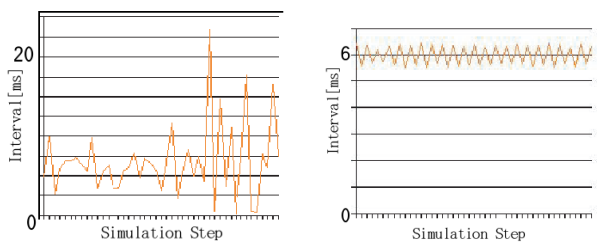


図 3 視覚情報提示間隔 (左: シミュレーションキャッシングなし, 右: シミュレーションキャッシングあり)

Fig. 3 Output interval of simulation result (Left: w/o simulation caching, Right: w/ simulation caching).

して存在し、通信路の遅延変動(ジッタ)による通信時間の変動も発生する。この問題を解決するため、リモートサーバで実行中のシミュレーションの一部をローカルサーバ上でも並列に冗長実行し操作端末に結果をストックする。これによりユーザからのインタラクションに対してリモートサーバが即応できない場合でもローカルサーバのストックデータで即応可能である(図 2)。2つのサーバの連携により予測不可能な通信遅延の変動による影響を隠蔽し実時間インタラクションを可能にする手法である。図 3 は、シミュレーションキャッシングの有無による結果提示間隔の違いを示すものである[7]。

現在の高性能プロセッサに必要な不可欠なキャッシュ・メモリが主記憶上のデータの一部をキャッシュしメモリアクセス遅延を隠蔽するのにに対して、シミュレーションキャッシングではあたかも計算の一部をキャッシングしているかのように振る舞う。

3.2.2 リモート・ローカルサーバ間連携法

リモートサーバとローカルサーバ間の連携法は様々な形態が考えられる。たとえば計算結果の提示において、出力データが精度に対して厳しい制約がある場合は多少遅延があったとしてもリモートサーバの結果を使用し、逆に応答時間に厳しい制約がある場合は多少精度が落ちたとしてもローカルサーバの結果を提示するといった方法がある。その他の連携法の例を以下に示す。

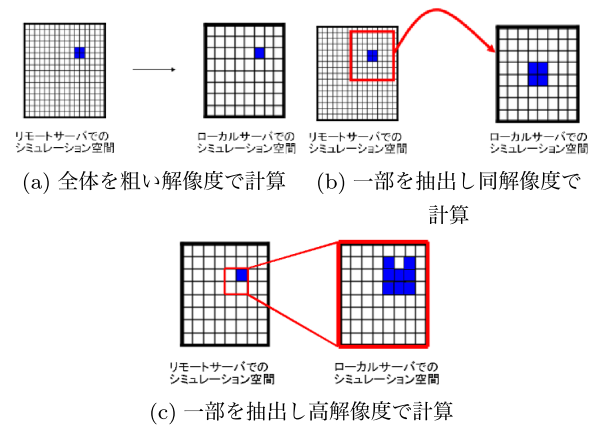


図 4 リモート・ローカルサーバ間連携法

Fig. 4 Remote and local server's collaboration schemes.

- リモートサーバで計算するシミュレーション領域全体を粗い解像度でシミュレーションする(図 4(a))。シミュレーション領域が 2 次元の場合、解像度が縦横 $1/N$ とすると計算量は $1/N^2$ となる。時間方向の計算量も $1/N$ となり全体として $1/N^3$ となる場合もある。
- シミュレーションする領域を限定し、リモートサーバで行っている領域の一部についてシミュレーションを行う(図 4(b))。縦横 $1/N$ の領域を選択しシミュレーションする場合、計算量は $1/N^2$ となる。
- シミュレーション領域の一部を高解像度でシミュレーションする(図 4(c))。注目領域内においてはリモートサーバよりも高精度のシミュレーションを行う。
- 異なる数値計算モデルを使用する。打ち切り誤差を大きくするなど、計算精度とトレードオフの関係にある計算モデルやアルゴリズムを使用する。
- 異なる数値精度を使用する。リモートでは倍精度で計算しているところを、単精度実数を用いる。

これらは適用するシミュレーションの内容および計算環境を加味したうえで決定する必要がある。

3.2.3 一貫性制御

シミュレーションキャッシングを実行するとき、シミュレーションが時間的な依存関係を持つ場合、シミュレーションが進むにつれて各サーバでの計算結果の違いが大きくなるという問題が発生する。そのため、シミュレーションがある程度進行した時点でシミュレーション精度の高いサーバのシミュレーション内容をもう片方のサーバに送信し反映する。これを一貫性制御と呼び、これまでに中断モデルとロールバックモデルの 2 つを提案している。

中断モデルは、一貫性制御が終了するまで入力を一時中断し、データの一貫性が保証されるまでの間シミュレーションをストップさせ、一貫性制御データの反映が終わってからリスタートする(図 5)。ロールバックモデルはシミュレーションを継続したまま一貫性制御を見かけ上同時並列して実行し、タイムステップが一致した時点でデータ

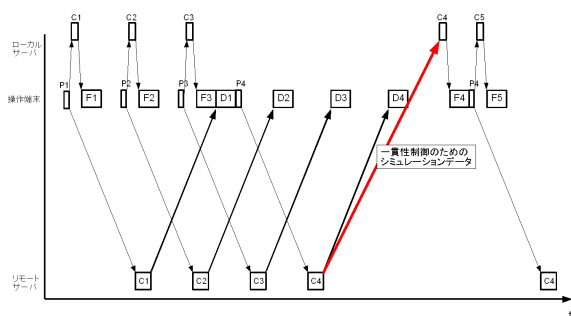


図 5 中断モデル

Fig. 5 Interruption model.

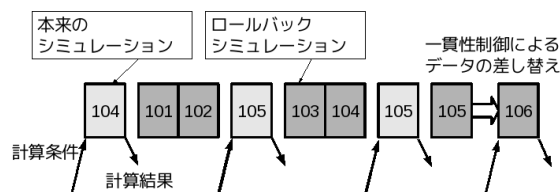


図 6 ロールバックモデル

Fig. 6 Rollback model.

を更新する(図6)。中断モデルで一貫性制御は、シミュレーションが一時中断してしまうため、応答時間が大きい場合や頻繁に一貫性制御を行う場合不向きであるが、一貫性制御のタイミングを確実に保証する。これに対してロールバックモデルは、シミュレーションと同時に進行して過去の計算条件を利用し、一貫性制御を行うためシミュレーションが中断することを避けられるが、ローカルサーバでの計算量が増え、明確にどのタイミングでデータの一貫性が保証されるか不定である。

一貫性制御は、対象とするシミュレーション内容によって送信するデータの形式(シミュレーション結果データ、圧縮したデータ、可視化画像など)や制御モデル、一貫性制御実施の頻度をカスタマイズする必要がある。

4. シミュレーションキャッシングフレームワーク

4.1 要件・実装方針

シミュレーションキャッシングを実際のシミュレーションに対して適用する場合、適用するシミュレーションのコードをベースにシミュレーションキャッシング部分を追加する必要がある。しかし、シミュレーションキャッシングでは、遠隔操作と各サーバの連携(遅延補償/一貫性制御)に関する記述が必要である。本来のシミュレーションとは無関係な、ネットワーク関連のコードやデータ構造の追加など、煩雑な処理が多くなり開発効率の低下を招く。シミュレーションキャッシングを適用する場合、シミュレーションキャッシング部分と対象とするシミュレーション部分の依存関係は低く、シミュレーションの内容が変更された場合でも、シミュレーションキャッシング部分では

パラメータの変更などの最小限の変更ですむ場合が多い。

そこで、シミュレーションキャッシングを実現するために必要な機能をフレームワークとして提供することを提案する。その場合ユーザは本来のシミュレーション部分や入出力処理を行う部分の記述へ注力することに専念できるため、システムの開発効率の向上が期待できる。フレームワークを用いることで必要となる専門知識も最低限で済み、容易にシミュレーションキャッシングを適用した対話型遠隔シミュレーションを実行できる。

4.1.1 想定するシステム構成

想定するシステムのハードウェア構成は、3.2節で述べた操作端末・リモートサーバ・ローカルサーバの3つをネットワークで結んだものを想定する。また、近年の同一端末での並列実行性の高さを受けて、操作端末とローカルサーバを同一端末として実行し、操作端末兼ローカルサーバ・リモートサーバの2つをネットワークで結んだ構成にも容易に移行できるよう配慮する。

4.1.2 機能要件

シミュレーションキャッシングを実現するためには次のような機能を持ったものが最低限必要となる。なお、遠隔操作の実現に関わる機能を(A)、サーバ間連携に必要な機能を(B)で表したときの対応関係も以下に示す。

- 操作端末から各サーバへの計算条件の送受信 (A)
- 各サーバから操作端末への計算結果の送受信 (A)
- パイプライン実現のための時間管理 (A) (B)
- 一貫性制御処理 (B)
- 操作端末における計算結果の自動選択 (B)
- 各プロセス間での同期 (B)
- スロースタートアルゴリズム対策 (B)

スロースタートアルゴリズム(以下SSA)とは、TCP/IPで輻輳制御に用いられるアルゴリズムで、通信が確立した直後のネットワーク通信に対して帯域幅が制限されるというものである。一貫性制御の時間間隔が大きくなると、一貫性制御が行われる前にSSAが適用され、通信速度が低下してしまう。特に一貫性制御のデータサイズが小さい場合問題となり、何らかの対策が必要である。

これらの機能のうち、遠隔操作に関する機能(A)は、シミュレーションキャッシング技術使用の有無にかかわらず、遠隔シミュレーションの実現に必要なものである。そのためサーバ間連携に関する機能(B)の提供がフレームワークの主目的であるが、(A)の通信処理に関する機能もフレームワークに一元化することでユーザの負担軽減を図る。

4.1.3 実装方針

様々なシミュレーションに対応でき、ユーザの負担を軽減するため、以下の方針に従って実装を行う。

- 通信内容などに制限を加えないようにする。
 - 異なる実行環境への移植性の確保。
- (変更箇所局所化、極小化)

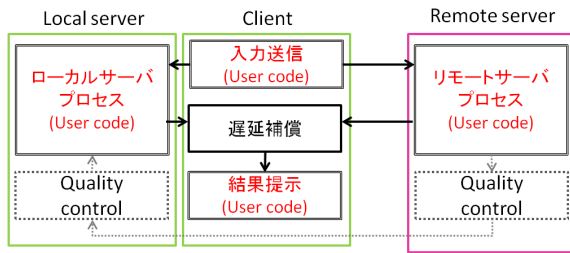


図 7 シミュレーションキャッシングフレームワーク

Fig. 7 Simulation caching framework.

4.2 仕様・システム設計

C 言語を想定し、ユーザエンドの入出力処理には OpenGL のフレームワークを利用し、それ以外の各機能を関数として提供する。関数として提供することが難しい機能に関しては、ユーザが作成するプロセスとは別にシミュレーションキャッシングフレームワーク専用のプロセスを設置する。遠隔操作の実現に関わる通信機能はすべて関数として提供する。しかし、時系列シミュレーションのパイプライン化や遠隔環境への適応を考え、入力送信プロセスと結果提示プロセスを個別のプロセスとして実装し各サーバと通信を行う。一貫性制御に必要な通信機能はすべて関数で提供する。しかし、一貫性制御データが比較的小さい場合 SSA 対策を行い通信路のクオリティを維持する必要があるため、各サーバにクオリティコントロールプロセスを実装し、一貫性制御と SSA 対策を行う。遅延補償に必要な機能は関数として提供することが難しいため、専用の遅延補償プロセスを設置する。各プロセス間の通信には MPI を使用する。全体の構成を図 7 に示す。

シミュレーションキャッシングに使用するパラメータはヘッダファイルの形で提供され、ヘッダファイルを書き換えることでシミュレーションキャッシングの様々なパラメータを変更することができる。シミュレーションキャッシングのパラメータは、1) パイプラインピッチなどアプリケーションと実行環境に依存するパラメータ、2) フレームワーク内のバッファサイズなど実行環境に依存するパラメータ、3) 一貫性制御間隔などアプリケーションに依存するパラメータの 3 種類がある。フレームワークとして提供する関数群は、1) フレームワークの初期化や終了処理などフレームワークを利用する際に 1 度だけ呼び出す関数、2) 計算条件や計算結果の送受信に関する毎ステップ呼び出される関数、3) 一貫性制御に関する処理など特定の条件時のみ処理を行う関数の 3 種類がある。一貫性制御に関する関数は各モデルおよび機能によって使用する関数が異なる。

4.3 各プロセスの機能

4.3.1 入力送信プロセス

操作端末の入力送信プロセスは、各サーバプロセスへ送信する対話的な入力条件の準備とパイプライン化のための

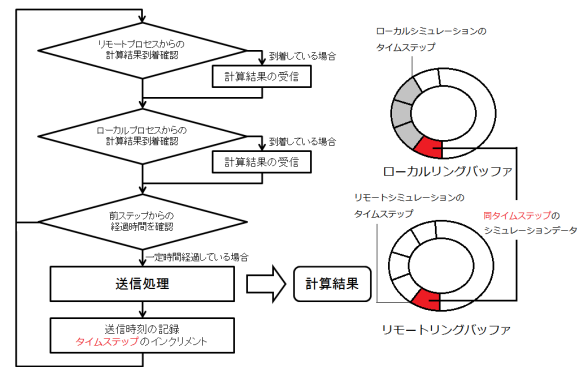


図 8 遅延補償プロセスの概要

Fig. 8 Overview of jitter compensation process.

各サーバプロセスへの一定間隔の計算条件送信が主な処理となる。フレームワークではこのうち、計算条件の一定間隔送信機能を関数として提供する。また、中断モデルを用いた一貫性制御時に、一貫性制御が終わるまで計算条件の送信を止める機能も関数として提供する。ユーザは自身の行いたいシミュレーションの計算条件をここへ記述する。

4.3.2 結果提示プロセス

出力プロセスは、遅延補償プロセスから結果を受け取り、出力をユーザに提示することが主な処理となる。フレームワークでは、結果の取得機能を関数として提供する。この関数では、どちらのサーバからの結果を受け取ったかという情報と受け取ったデータサイズを知ることができ、ユーザはこの情報をもとにして異なる処理（補間処理など）を実行可能である。

4.3.3 リモート・ローカルサーバプロセス

リモート・ローカルの各サーバプロセスは、計算条件の受信、シミュレーションの実行、結果データの可視化（任意）、結果の送信、一貫性制御が主な処理となる。このうち、フレームワークではシミュレーション以外の処理をそれぞれ関数として提供する。特に一貫性制御に関しては、使用するモデルに応じて複数の関数を提供し、ユーザは使用するモデルによって関数を選択することで一貫性制御を実現する。

4.3.4 遅延補償プロセス

遅延補償プロセスは、3.2.1 項で述べたシミュレーションキャッシングにおける通信遅延のゆらぎを補償するプロセスである。このプロセスは、各サーバプロセスから非同期で送信される計算結果の受信、受信した結果のシミュレーションステップによるバッファリング、一定間隔での結果提示プロセスへの計算結果送信が主な処理となる。図 8 に処理の流れと結果データ格納用の構造体を示す。使用するバッファの構造はリングバッファで、リモート・ローカル用に 2 つ用意する。送信元のサーバプロセス識別番号、シミュレーションタイムステップおよびデータ格納先ポインタを持ち、データ格納領域サイズとバッファのエントリ

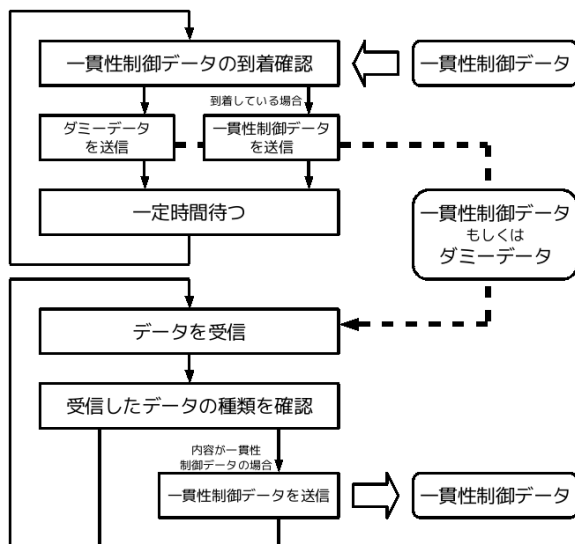


図 9 クオリティコントロールプロセスの概要

Fig. 9 Overview of quality control process.

数はヘッダファイルで変更できる。どちらかのサーバのシミュレーションが先行しバッファに蓄積したとしても、格納時に付加した情報をもとに結果データの比較・送信が可能である。送信終了後、リードポインタを進めデータを破棄する。

4.3.5 クオリティコントロールプロセス

クオリティコントロールプロセスでは、一貫性制御の実施と SSA 対策が主な処理となる。リモートサーバからの計算結果送信の間隔と比べて、一貫性制御の間隔は大きく（たとえば 100 倍）設定する。そのため、リモートサーバとローカルサーバ間に十分な通信帯域があるにもかかわらず、SSA の影響を受け必要以上の通信遅延が発生する可能性がある。そこで一貫性制御を行う必要のない時刻においても、定期的にクオリティコントロールプロセス間で数バイトのデータを送出することで、通信遅延の増加を回避している。図 9 に処理の流れを示す。ただし、一貫性制御データが大きい場合や一貫性制御間隔が小さい場合、SSA の影響は相対的に小さいものと思われるため、このプロセスを使用するか、使用せず直接リモートサーバプロセスとローカルサーバプロセスで通信して一貫性制御するかをユーザが簡単に変更できる仕様になっている。

4.4 一貫性制御モデルの実装

一貫性制御では、使用する一貫性制御モデルに応じて実行する処理が異なる。フレームワークでは、3.2.3 項で述べたモデルに応じた関数をそれぞれ提供する。ユーザは適用する一貫性制御モデルに応じてこれらの関数を選択し一貫性制御を行う。

中断モデルでは一貫性制御時、まず入力送信プロセスが停止する。次にリモートサーバプロセスはクオリティコントロールプロセスを経由して、ローカルサーバプロセスへ

一貫性制御データを送信する。このとき送信される一貫性制御データはユーザの指定がない限りリモートサーバプロセスで扱っている結果データと同じデータである。ローカルサーバプロセスは受信した一貫性制御データを自プロセスの計算結果データに反映させた後、入力送信プロセスへシミュレーション再開のメッセージを送り一貫性制御を終了する。

ロールバックモデルでは、リモートサーバプロセスからクオリティコントロールプロセスを経由してローカルサーバプロセスへ一貫性制御データを送信する。ローカルサーバプロセスはシミュレーションを継続させたまま、並行して実行するロールバックシミュレーションへ一貫性制御データを反映させ、ローカルサーバのタイムステップに追いついた時点で計算結果データをローカルサーバにコピーし一貫性制御を終了する。中断モデルと違い、入力プロセスでは計算条件の送信は通常どおり行われるので特別な処理は必要ないが、ロールバックシミュレーションをユーザが記述する必要がある。

4.5 フレームワークが提供する関数

遠隔操作と一貫性制御の実現に関わる通信機能はすべて関数として提供する（図 10）。以下に各関数の使用法とその機能を示す。

4.5.1 シミュレーションキャッシングの初期化処理

SC_INITIALIZE は、シミュレーションキャッシングに必要な初期化を行う関数で、各プロセスのはじめに 1 度だけ利用する。第 1 および第 2 引数ではコマンドライン引数の数や文字列を指定する。第 3 引数には各ランクの番号が割り振られる。ランク番号とは、実行するプロセスの各々に与えられる固有の番号で、このランク番号を使用することでプロセスごとに機能を変更できる。第 4 引数には全体のプロセス数が格納される。正常終了した際には 0 が返される。

4.5.2 パイプライン先頭の同期

シミュレーションパイプラインでは、パイプラインの先頭において各プロセスでシミュレーションの準備ができている必要がある。パイプラインのスタートの前には各プロセスにおいて初期化処理があり、初期化処理の内容は各プロセスやアプリケーションの内容に依存し一定ではない。そこで、パイプラインをスタートする直前で各プロセス間で同期を取り、準備ができていることを確認する。SC_READY は、各プロセス間で同期を取りパイプラインの先頭を揃える関数であり、各プロセスの初期化などの処理が終わった後 1 度だけ利用する。正常終了した際には 0 が返される。

4.5.3 シミュレーションキャッシングの終了処理

SC_FINALIZE では、シミュレーションキャッシングの終了処理およびプロセス間通信に使用した MPI の終了処

```

(a) シミュレーションキャッシングの終了処理
int SC_INITIALIZE
(int *argc, //コマンドライン引数の数
char **argv, //コマンドライン引数の文字列
int *proc_num, //プロセス数格納変数
int *rank) //ランク番号格納変数

(b) バイブライン先の同期
int SC_READY(void)

(c) シミュレーションキャッシングの終了処理
int SC_FINALIZE
(int rank) //プロセス番号

(d) 計算条件の登録
int SC_INPUTSET
(void *message, //送信するデータの先頭ポインタ
int count, //送信するデータの要素数
int datasize, //1要素のデータサイズ [Byte]
int step) //シミュレーションステップ番号

(e) 計算条件の取得
int SC_INPUTGET
(void *buffer, //受信するバッファの先頭ポインタ
int count, //受信するデータの要素数
int datasize, //1要素のデータサイズ [Byte]
int step) //シミュレーションステップ番号

(f) 計算結果の登録
int SC_OUTPUTSET
(void *message, //送信するデータの先頭ポインタ
int count, //送信するデータの要素数
int datasize, //1要素のデータサイズ [Byte]
int step) //シミュレーションステップ番号

(g) 計算結果の取得
int SC_OUTPUTGET
(void *buffer, //受信するバッファの先頭ポインタ
int *count, //受信したデータの要素数
int datasize, //1要素のデータサイズ [Byte]
int step) //送受信するデータのステップ番号

(h) 一貫性制御 中断モデル
void SC_INTEGRITYCONTROL_S
(void *data, //一貫性制御データの読み書き先
int count, //送信するデータの要素数
int datasize, //1要素のデータサイズ [Byte]
int step, //シミュレーションステップ番号
int rank, //ランク番号
int data_no, //一貫性制御データ ID
int data_num) //一貫性制御データの種類の数

(i) 一貫性制御 入力停止
void SC_INTEGRITYCONTROL_I
(int step, //シミュレーションステップ番号
int data_num) //一貫性制御データの種類の数

(j) 一貫性制御 ロールバックモデル
int SC_INTEGRITYCONTROL_R
(void *sim_data, //シミュレーションデータポインタ
void *roll_data, //ロールバックデータポインタ
int count, //送受信するデータの要素数
int datasize, //1要素のデータサイズ [Byte]
int step, //シミュレーションステップ番号
int *roll_step, //ロールバックステップ番号
int rank, //ランク番号
int data_no, //一貫性制御データ ID
int data_num) //一貫性制御データの種類の数

(k) 一貫性制御 チェック
int SC_INTEGRITYCONTROL_CHECK
(int step) //シミュレーションステップ番号

```

図 10 フレームワーク関数群

Fig. 10 Functions provided by our framework.

理を行う。各プロセスの終了直前で使用する。正常終了した際には0が返される。

4.5.4 計算条件の登録

SC_INPUTSET は、指定されたデータサイズのデータを各シミュレーションサーバに送信する関数で、入力送信プロセスで使用する。また、前回からの呼び出しからの経過時間がパイプラインピッチより短い場合、送信間隔が一定となるよう時間を調整した後各サーバに送信する。正常終了した際には0が返される。

4.5.5 計算条件の取得

SC_INPUTGET では、入力送信プロセスから入力データを受け取り、第1引数で指定されたバッファ領域に計算条件を書き込む。それと同時に内部にあるリングバッファにステップ番号と送られてきた計算条件を格納する。第4引数で与えられるシミュレーションステップ番号が過去のステップ番号である場合、リングバッファを参照し指定されたタイムステップの計算条件を第1引数で指定されたバッファ領域に書き込む。過去の入力条件は、ロールバックモデルを採用した場合に必要となる。正常終了した際には0が返され、バッファにないステップ番号の結果の要求があった場合は-1が返される。この関数は、各サーバプロセスで使用する。

4.5.6 計算結果の登録

SC_OUTPUTSET では、指定されたデータサイズのデータを操作端末に送信する。正常終了した際には0が返される。この関数は、各サーバプロセスで使用する。

4.5.7 計算結果の取得

SC_OUTPUTGET は、第4引数で指定されたシミュレーション番号の結果を第1引数で指定されたバッファ領域に格納する。引数は先の3つの関数と似ているが、要素数はポインタ渡しとし、ここには受信された要素数が格納される。これは、ローカルサーバの結果を受信した場合とリモートサーバの結果を受信した場合でデータサイズが異なるケースや、シミュレーションステップごとに一定のデー

タサイズの結果ではなく前ステップからの差分だけを送る場合など、シミュレーションステップごとにデータ量が増えるケースにも対応するためである。返り値からどちらのサーバから受信したのかが分かる。これら情報を使用することで、受信したデータの送信元やデータサイズによって処理を変更することができる。

4.5.8 一貫性制御に関する関数群

SC_INTEGRITYCONTROLS は、各プロセスにおいて一貫性制御を行う関数である。第1引数には、リモートサーバ側では送信する一貫性制御データが格納されているデータ領域の先頭アドレスを指定し、ローカルサーバ側では一貫性制御データを適用するデータ領域のアドレスを指定する。一貫性制御が必要なタイムステップの場合には、リモートサーバの領域に格納されているデータをローカルサーバに送信し、ローカルサーバの領域に上書きする。よって、第2引数のデータの数および第3引数のデータの種類の数は共通のものを使用する。第4引数では現在のシミュレーションタイムステップを指定する。関数内部では、このステップ番号によって一貫性制御を行うかを判断している。第5引数では、呼び出すプロセスの役割 (REMOTEorLOCAL) を与える。これによって、送信処理を行うか受信処理を行うかを決定する。第6・7引数では一貫性制御データを識別するIDと、全体で何種類の一貫性制御データが必要かを指定する。この引数を調節することで一貫性制御データが複数ある場合に対応できる。ユーザコードで複数のデータを1つにまとめて一貫性制御を行うことも可能である。また、中断モデルを使用する場合、入力送信プロセスはローカルサーバの一貫性制御の終了を待つ必要がある。フレームワークでは、入力送信プロセスでローカルサーバプロセスの一貫性制御処理の終了を待つ関数を用意している。

SC_INTEGRITYCONTROLI では、第1引数としてタイムステップを入力することで、そのタイムステップで一貫性制御が必要かを判定しており、一貫性制御を行う必要

がない場合はすぐに制御が戻る。第2引数では一貫性制御に使用されているデータの個数を指定する。ローカルサーバプロセスの `SC_INTEGRITYCONTROLS` では、一貫性制御データの適用が終了した後に終了メッセージを入力プロセスに送信している。`SC_INTEGRITYCONTROL_I` はその終了メッセージを受信するまで待機し、関数から制御が戻ることでローカルサーバプロセスの一貫性制御の終了を保証する。中断モデルで一貫性制御を行う場合は `SC_INTEGRITYCONTROLS` を使用すればよいが、ロールバックモデルを使用する場合はローカルサーバプロセスにおいて、代わりに `SC_INTEGRITYCONTROL_R` を使用する。これは、中断モデルの場合は受信したデータをそのまま反映させればよいのに対し、ロールバックモデルでは現在のシミュレーションステップと受信した一貫性制御のシミュレーションステップが異なるので、必要な機能が異なるためである。リモートサーバプロセスは使用するモデルにかかわらず処理は共通なので `SC_INTEGRITYCONTROLS` を使用すればよい。

`SC_INTEGRITYCONTROL_R` は、一貫性制御としてロールバックモデルを使用する際、ローカルサーバプロセスで使用する。第1引数および第2引数には本来のシミュレーションデータが格納されている領域と、フィードバック作業を行っている領域を指定する。ロールバック作業が完了した場合、シミュレーションデータ領域にロールバックされた一貫性データが上書きされる。第3～5引数はそれぞれ、`SC_INTEGRITYCONTROLS` の `count`, `datasize`, `step` と同じものを使用する。第6引数は、ロールバックを行うためのタイムステップ変数をポインタで指定する。第7～9引数はそれぞれ、`SC_INITIALIZE` 関数で取得するプロセス番号、複数の一貫性制御データで一貫性制御を行う際の番号と数を指定する。

`SC_INTEGRITYCONTROL_CHECK` はどのプロセスでも使用できるが、主にリモートサーバプロセスで使用されることを想定している。引数は、現在のシミュレーションタイムステップを指定する。この関数の返り値は、一貫性制御が行われる場合は0、行われない場合は1である。返り値を使用して一貫性制御が行われる場合のみ特定の処理を実行することが可能となる。

5. 評価

福井大学の我々の研究室にローカルサーバ兼操作端末、北海道大学情報基盤センターのプロジェクトサーバにリモートサーバを設置した。なお、学内ネットワーク環境の制約のため学外に商用のVPNサーバをレンタルし、このサーバを介して福井大学－北海道大学間でシミュレーションキャッシングを実行する環境を構築した。このときの福井大学－北海道大学間の通信帯域は約7[Mbps]であり、通信遅延の平均は30[msec]であった。表1に実験環境を示

表1 実験環境

Table 1 Experiment environment.

	リモートサーバ	ローカルサーバ兼操作端末
CPU	IntelXeon E7-8870 2.4 GHz	IntelCorei7 940 2.93 GHz
コア数	10	4
メモリ	30 GB	1 GB
OS	Linux 32 bit	Linux 32 bit
MPI	MPICH2-1.1	MPICH2-1.1

す。アプリケーションへのフレームワークの適用試験、フレームワークの機能評価、SSA対策効果の確認のため実験を行った。

5.1 シミュレーションへの適用

フレームワーク化の効果を確認するため、実際に我々が所有している複数のリアルタイムシミュレーションアプリケーションに本研究のフレームワークを適用し、シミュレーションキャッシングを用いて遠隔シミュレーションの実験を行った。以下の3つのアプリケーションは遠隔実時間ステアリングに対応していなかったため、入力送信プロセスに入力を生成するコードを追記し、シミュレーション本体では外部との入出力が可能になるよう変更を行った。可視化と表示部については元アプリケーションのコードをそのまま移植し通信部のみフレームワークのフォーマットに合わせ実装した。サーバ間連携法は図4(a)、一貫性制御は中断モデルを使用した。ステアリング部・シミュレーション本体の修正・通信関数などの新たにユーザが記述したコードは20行ほど（ヘッダファイルの変更などは除く）であり、容易にフレームワークの適用ができることが分かる。付録にフレームワークを適用した熱拡散シミュレーションのソースコードの一部とシミュレーションキャッシング用のヘッダファイルを添付する。simulationと呼び出している関数が本来のシミュレーションコードである。

5.1.1 熱拡散シミュレーション

熱伝導方程式ベースの差分法による熱拡散シミュレーションを使用した。このシミュレーションでは領域内に動く熱源を用意し、それに応じて領域内の温度分布がどのようなかをシミュレーションする。入力送信プロセスから各サーバプロセスに送信される計算条件は領域内の熱源座標(XY座標)、結果提示プロセスで受信するデータは、領域内の温度分布を可視化したものとした。通信データサイズは入力座標が16[Byte]、シミュレーション結果の可視化データが48[KB]、一貫性制御データが128[KB]である。シミュレーション領域は二次元で、リモートサーバでは128x128の解像度、ローカルサーバでは、領域全体をリモートサーバの1/2の解像度(64x64)でシミュレーションを実行した。ローカルサーバでの計算量は時間方向も含め1/8である。シミュレーションの様子を図11(a)に示す。

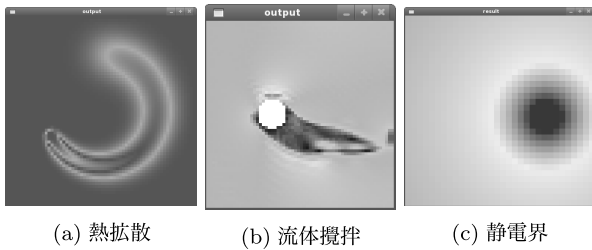


図 11 シミュレーション結果のスナップショット例
Fig. 11 Snap shot example of simulation results.

5.1.2 流体シミュレーション

格子ボルツマン法による流体攪拌シミュレーション [7] を使用した。このシミュレーションでは、円柱を流体の中に設置し、その円柱を動かすことによる周囲の流速の変化をシミュレーションする。入力送信プロセスから各サーバプロセスに送信される計算条件は領域内の円柱座標 (XY 座標)、結果提示プロセスで受信するデータは、領域内の流体速度を可視化したものとした。通信データサイズは入力座標が 16 [Byte]、シミュレーション結果の可視化データが 48 [KB]、一貫性制御データが 128 [KB] である。シミュレーション領域は二次元で、リモートサーバでは 64x64、ローカルサーバでは 32x32 の解像度でシミュレーションを行う。シミュレーションの様子を図 11 (b) に示す。なお、文献 [7] のプログラムでは視覚・力覚提示を行っていたが、今回の設定では力覚提示は省略している。

5.1.3 静電界シミュレーション

領域内に点電荷を設置し、その周囲の電界の変化のシミュレーションを使用した。入力送信プロセスから各サーバプロセスに送信される計算条件は領域内の点電荷座標 (XYZ 座標)、結果提示プロセスで受信されるデータは、領域内の電界強度を可視化したものとした。通信データサイズは入力座標が 32 [Byte]、シミュレーション結果の可視化データが 12 [KB]、静電場のため一貫性制御は行う必要はない。シミュレーション領域は三次元、リモートサーバでは 64x64x64、ローカルサーバでは 32x32x32 の解像度でシミュレーションを行う。また、結果提示プロセスで受信する各サーバプロセスからの可視化データは 32x32 の二次元電界強度とし、領域内の指定された平面の結果を画面に出力する。シミュレーションの様子を図 11 (c) に示す。

5.2 性能評価

フレームワークの機能であるジッタ隠蔽効果と一貫性制御の影響を調べるため、熱拡散シミュレーションを用いて評価実験を行った。遅延補償プロセスの結果データ選択時にリモートサーバの結果データが間に合わずローカルサーバからの結果データを選択した割合 (許容遅延違反率)、シミュレーションステップごとの遅延補償プロセスから結果提示プロセスへのデータ送信間隔 (結果データ提示間隔)

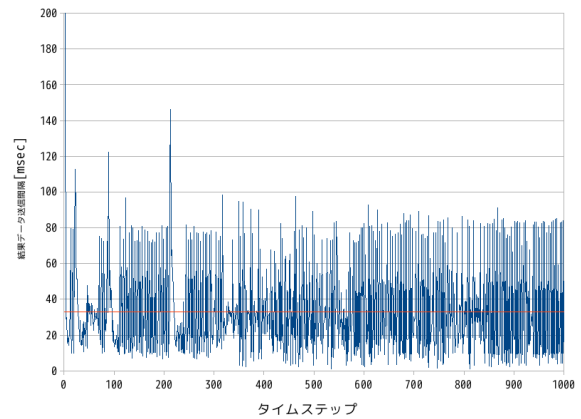


図 12 熱拡散シミュレーションの結果データ提示間隔 (シミュレーションキャッシングなし)
Fig. 12 Output interval in heat distribution simulation (w/o simulation caching).

表 2 許容遅延違反率と実行時間

Table 2 Output interval violation ratio and execution time.

試行 No.	1	2	3	4	5
許容遅延違反率 [%]	57.7	61.0	29.5	45.8	43.1
1000step 実行時間 [sec]	35.5	35.8	36.2	36.5	36.0

の測定を行った。シミュレーション単体での 1 ステップにかかる時間はリモートサーバ上で約 1 [msec] である。パイプラインピッチは、実験環境デバイスの表示 FPS の上限、人間が違和感なく可視可能な値として 30 [fps] (33 [msec]) に設定した。比較対象としてシミュレーションキャッシングなしでパイプライン化した熱拡散シミュレーションをリモートサーバで 1000 ステップ実行した場合の各ステップにおける結果データ提示間隔を図 12 に示す。このときのパイプラインピッチは 33 [msec]、許容遅延違反率は 33.4 [%] で全体の処理時間は 33.63 [sec] である。シミュレーションキャッシングを行っていないため、3 割ほどのデータが遅れてユーザに提示されている。提示間隔のゆらぎを視覚的にもはっきりと感ずることができる。

5.2.1 フレームワークのジッタ隠蔽効果

フレームワークのジッタ隠蔽機能により、設定した間隔での結果提示ができるかを確認する。一貫性制御間隔を 100 ステップに設定し、異なる 5 回の試行を行った。許容遅延違反率とプログラム全体の実行時間を表 2 に、5 回の試行のうち遅延違反率が図 12 と最も近い 29.5 [%] と違反率が最も高い 61.0 [%] の 1000 ステップ間の結果データ提示間隔を図 13、図 14 に示す。表 2 から、同じシミュレーション環境でも通信路の遅延変動の影響を受け許容遅延違反率が大きく変動することが分かる。しかし、フレームワークのジッタ隠蔽機能により、図 13、図 14 のように設定したパイプラインピッチで結果提示できている。一定間隔でパイプラインピッチを超える時間が測定されているが、これは中断モデルの一貫性制御によるもので、中断モ

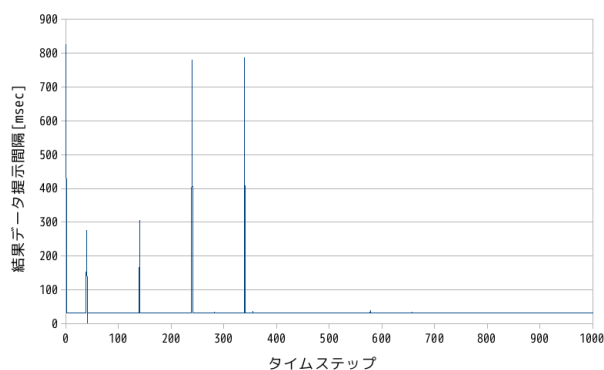


図 13 結果データ提示間隔 (違反率 29.5 [%])

Fig. 13 Output interval (Violation ratio 29.5 [%]).

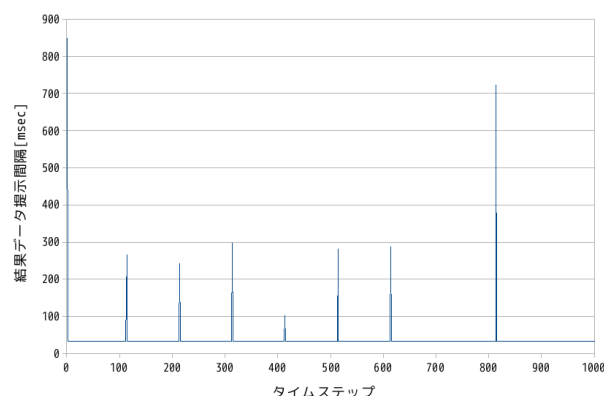


図 14 結果データ提示間隔 (違反率 61.0 [%])

Fig. 14 Output interval (Violation ratio 61.0 [%]).

デルでは一貫性制御が終わってリモートサーバの結果が到着するまで遅延補償プロセスを停止しているためである。

シミュレーションキャッシングなしの実験結果 (図 12) と比較すると、パイプラインピッチを超える時間を要するのは初期化と一貫性制御のステップのみであり、それ以外のステップでは結果提示間隔のゆらぎは感じられない。

5.2.2 一貫性制御間隔の影響

一貫性制御を実施する間隔を変更した場合のシミュレーション全体への影響を調べるため一貫性制御間隔を変化させて測定を行った。許容遅延違反率とプログラム全体の実行時間を表 3、一貫性制御間隔が 25 ステップと 250 ステップのときの 1000 ステップ間の結果データの提示間隔を図 15、図 16 に示す。一貫性制御のタイミングで遅延が見られない箇所もあり、一貫性制御を実施した際に遅延が発生するか否か、発生した遅延の大小などは通信路の輻輳状態に依存する。一貫性制御間隔が短くなると、全体の実行時間や結果データの提示間隔に影響が出る。これは一貫性制御間隔が短くなることで、入力送信・ローカルサーバプロセスが頻繁に待機状態になり、パイプライン化の効果が十分に発揮されないためである。逆に、一貫性制御間隔が長すぎるとローカルサーバでの計算誤差が次第に蓄積し、そのような状況でローカルサーバの結果が提示されると提

表 3 一貫性制御間隔を変更した場合の許容遅延違反率と実行時間

Table 3 Output interval violation ratio and execution time for different consistency control period.

一貫性制御間隔 [step]	25	50	250	500
許容遅延違反率 [%]	73.6	75.1	29.0	14.0
1000step 実行時間 [sec]	39.4	38.5	34.9	34.0

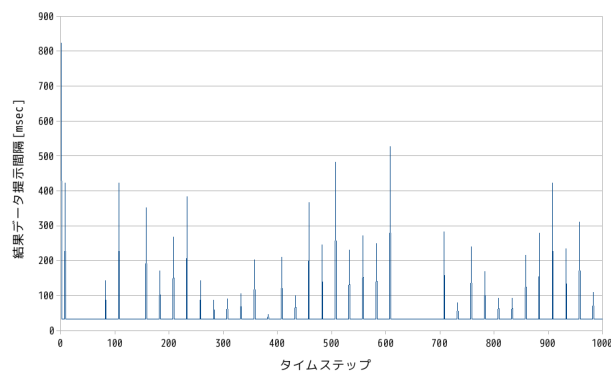


図 15 一貫性制御間隔が 25 ステップの結果データ提示間隔

Fig. 15 Output interval for consistency control period of 25 steps.

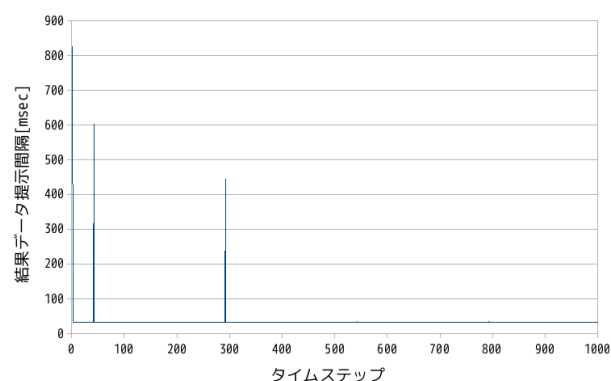


図 16 一貫性制御間隔が 250 ステップの結果データ送信間隔

Fig. 16 Output interval for consistency control period of 250 steps.

示画像にちらつきが感じられる。画像処理でちらつきを軽減することは可能であるが、ローカルサーバの誤差を軽減するためには一貫性制御の間隔を短くする必要がある。遅延隠蔽効果と計算精度はトレードオフの関係にあり、ユーザは一貫性制御間隔の適切な設定が求められる。

5.3 クオリティコントロールプロセスの効果

4.3.5 項で述べた、SSA 対策の効果を調べる。熱拡散シミュレーションを用いて SSA 対策の有無で一貫性制御にかかる時間を比較する。一貫性制御間隔を 100 ステップに設定し、それぞれ異なる 4 回の試行にて測定を行った。各試行 1000 ステップのうち一貫性制御ステップ 10 回の平均時間を表 4、表 5 に示す。一貫性制御データが 32 [KB] と小さいときは SSA 対策により、3 倍近い高速化が確認できる。一貫性制御データが 128 [KB] の場合でも、SSA 対策

表 4 平均一貫性制御時間 (一貫性制御データ 32 [KB])

Table 4 Average time spent in consistency control (Data size of 32 [KB]).

試行 No.	1	2	3	4	4 回平均
SSA 対策有 [msec]	39	30	52	67	47
SSA 対策無 [msec]	131	143	128	121	131

表 5 平均一貫性制御時間 (一貫性制御データ 128 [KB])

Table 5 Average time spent in consistency control (Data size of 128 [KB]).

試行 No.	1	2	3	4	4 回平均
SSA 対策有 [msec]	220	244	230	249	236
SSA 対策無 [msec]	240	264	248	228	241

の効果はわずかながら確認できる。

6. まとめ

本論文では、遠隔地の高性能計算サーバを用いた対話型シミュレーションに対する遅延隠蔽手法の 1 つであるシミュレーションキャッシングのフレームワーク実装を行った。これにより他のアプリケーションにシミュレーションキャッシングを適用可能となった。フレームワークではシミュレーションキャッシングに必要な複雑な遠隔通信処理やサーバ間の連携処理をプロセスや関数で提供しており、ユーザは専門的な知識がなくても他シミュレーションへ容易にシミュレーションキャッシングの適用が可能である。

実際の通信路で性能評価を行い、フレームワークの遅延隠蔽効果が確認された。しかし、このフレームワークを使用する際、一貫性制御間隔や許容遅延時間などをシミュレーションやサーバ連携法に合わせて設定しなければならず、これはユーザにとって難しい問題となる。現状では、すべてユーザが定義する必要があるため、自動的にパラメータを生成する機能を追加していきたい。クオリティコントロールプロセスの SSA 対策については、今回の実験で扱ったシミュレーションのような小規模なものであれば効果があった。

また、本研究の発展としてシミュレーションキャッシングフレームワークを複数人で同時利用し、1 つのシミュレーション結果を利用者全員で共有し協調作業を可能にするシステムの研究も行っている [16]。

謝辞 本研究の一部は、JSPS 研究費 25280042 ならびに 25540043 の助成を得て実施した。本研究の一部は、北海道大学情報基盤センターのシステムを利用して実施した。また、数々の有力なご指導、ご意見をいただいた松山幸雄技術職員をはじめ、日頃様々な面でお世話になった本学森・福岡研究室の皆様へ深く感謝いたします。

参考文献

- [1] McCarty, W.D., Sheasby, S., Amburn, P., Stytyz, M.R. and Switzer, C.: A virtual cockpit for a distributed interactive simulation, *IEEE Computer Graphics and Applications*, Vol.14, No.1, pp.49–54 (1994).
- [2] Marshall, R., Kempf, J. and Dyer, S.: Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie, *Proc. Symp. on Interactive 3D Graphics*, pp.89–97 (1990).
- [3] Johnson, C., Parker, S.G., Hansen, C., Kindlmann, G.L. and Livnat, Y.: Interactive Simulation and Visualization, *IEEE Computer*, Vol.32, No.12, pp.59–65 (1999).
- [4] Zhu, Q. and Agrawal, G.: Supporting Fault-Tolerance for Time-Critical Events in Distributed Environments, *Proc. Supercomputing*, Paper No.32, pp.1–12 (2009).
- [5] Malakar, P., Natarajan, V. and Vddhiyar, S.S.: An Adaptive Framework for Simulation and Online Remote Visualization of Critical Climate Applications in Resource-constrained Environment, *Proc. Supercomputing*, Paper No.295, pp.1–10 (2010).
- [6] Woodward, P.R., Porter, D.H., Greensky, J. Larson, A.J., Knox, M., Hanson, J., Ravindran, N. and Fuchs, T.: Interactive Volume Visualization of Fluid Flow Simulation Data, *PARA'06 Workshop on the State-of-the-Art in Scientific and Parallel Computing* (2006).
- [7] 橋本健介, 手塚俊作, 森眞一郎, 富田眞治: シミュレーションキャッシングと遠隔インタラクティブ流体シミュレーションへの応用, 情報処理学会論文誌, Vol.5, No.4, pp.76–86 (2012).
- [8] 今村俊幸, 小出 洋, 武宮 博: 異機種並列計算機間通信ライブラリ Stampi 利用手引書 第二版, 日本原子力研究所 (2002).
- [9] 棟朝雅晴: 分散クラウドシステムにおける遠隔連携技術, 学際大規模情報基盤共同利用・共同研究拠点, 平成 25 年度共同研究最終報告書 (2014).
- [10] 實本英之, 小林泰三, 松本正晴, 滝澤真一郎, 三浦信一, 中島研吾: 複数拠点利用を実現するユーザ駆動型・拠点協調フレームワーク, 信学技報, pp.155–159 (2014).
- [11] 菅原章博, 岸本泰明: 大規模シミュレーションを中心に据えた遠隔研究システム II, *J. Plasma Fusion Res.*, Vol.87, No.3, pp.222–229 (2011).
- [12] 武井利文, 松本秀樹, 土肥 俊: リアルタイム可視化システム RVSLIB, 第 10 回計算力学講演会講演論文集, p.413 (1997).
- [13] 田村善昭, 小笠温滋, 中島拓之, 藤井孝藏: ライブラリレス・リアルタイム可視化システム, 可視化情報学会誌, Vol.25, No.Suppl.1, pp.251–254 (2005).
- [14] Waser, J., Fuchs, R. Ribicic, H., Schindler, B., Bloschl, G. and Groller, M.E.: World lines, *IEEE Trans. Visualization and Computer Graphics*, Vol.16, No.6 (2010).
- [15] Niebling, F., Becker, M., Kopecki, A., Stellba hydro GmbH & Co. KG, High Performance Computing Center Stuttgart (HLRS): Collaborative Steering and Post-Processing of Simulations on HPC Resource Everyone, Anytime, Anywhere, *Web3D* (2010).
- [16] 山本 優, 西村祐介, 福岡慎治, 森眞一郎: 対話型遠隔シミュレーションフレームワークのマルチクライアント拡張と予備評価, 情処研報, Vol.2014-HPC-147, No.15, pp.1–8 (2014).

付 録

A.1 フレームワークを適用した熱拡散シミュレーションのソースコードの一部

```
//入力送信プロセス
//評価用入力
void dataset(double *data, int step){
    double t = step * (2*M_PI/SPEED);
    data[0] = (DX/2) + 3.0*sin(t);
    data[1] = (DY/2) + 3.0*cos(t);
}
//メインループ
SC_READY();
for(step = 1; step <= MAXSTEP; step++){
    dataset(in_data, step);
    SC_INPUTSET(in_data, IN, sizeof(double), step);
    SC_INTEGRITYCONTROL_I(step, 1);
}

//ローカルサーバプロセス
//メインループ
SC_READY();
for(step = 1; step <= MAXSTEP; step++){
    //入力データ取得
    SC_INPUTGET(in_data, IN, sizeof(double), step);
    /*-----ユーザ記述部-----*/
    //シミュレーション
    simulation(out_data[0], out_data[1],
               in_data, 255.0, dt, N/2);
    //可視化データの作成
    convert_RGB(out_data[0], color, N/2);
    /*-----*/
    memcpy(out_data[0], out_data[1], sizeof(double)*OUT/4);
    //出力データ登録
    SC_OUTPUTSET(color, sizeof(color),
                 sizeof(GLubyte), step);
    //一貫性制御
    SC_INTEGRITYCONTROL_S(out_data[0], OUT/4,
                          sizeof(double), step, rank, 0, 1);
}

//リモートサーバプロセス
//メインループ
SC_READY();
for(step = 1; step <= MAXSTEP; step++){
    //入力データ取得
    SC_INPUTGET(in_data, IN, sizeof(double), step);
    /*-----ユーザ記述部-----*/
    //シミュレーション
    simulation(out_data[0], out_data[1],
               in_data, 255.0, DT, N);
    //可視化データの作成
    convert_RGB(out_data[0], color, N);
    /*-----*/
    memcpy(out_data[0], out_data[1], sizeof(double)*OUT);
    //一貫性制御データの作成
    makedata(mini_data, out_data[0], out_data[1], N, 2);
    //出力データ登録
    SC_OUTPUTSET(color, sizeof(color),
```

```
        sizeof(GLubyte), step);
    //一貫性制御
    SC_INTEGRITYCONTROL_S(mini_data, OUT/4,
                          sizeof(double), step, rank, 0, 1);
}

//結果提示プロセス
//メインループ
void idle(void)
{
    int size; // 計算結果サイズ
    int i;
    FILE* fp;
    int s;
    //シミュレーション結果を取得
    s = SC_OUTPUTGET(color, &size, sizeof(GLubyte), step);
    //再描画
    glutPostRedisplay();
    glFinish();
    //終了条件確認
    if(step == MAXSTEP){
        SC_FINALIZE(rank);
        exit(0);
    }
    step++;
}
```

A.2 シミュレーションキャッシング用ヘッダファイル

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<float.h>
#include<mpi.h>

//入力送信プロセスの送信間隔 [sec]
#define INTERVAL 0.033
//許容遅延時間 [sec] (INTERVAL とおなじ)
#define LIMIT_RELAY 0.033

//各プロセスのランク番号
//入力送信プロセス
#define INPUT 0
//結果提示プロセス
#define OUTPUT 1
//リモートサーバプロセス
#define REMOTE 2
//ローカルサーバプロセス
#define LOCAL 3
//遅延補償プロセス
#define RELAY 4
//クオリティコントロールプロセス (リモート)
#define QC_REMOTE 5
//クオリティコントロールプロセス (ローカル)
#define QC_LOCAL 6
//全体のプロセス数
#define PROC 7
```

```
//測定回数
#define MAXSTEP 1000
//入出力データ用領域
#define BUFSIZE_IN 500
#define BUFSIZE_OUT 500
//入出力・一貫性制御データの最大サイズ[Byte]
#define MAXBYTE_IN 100
#define MAXBYTE_OUT 140000
#define MAXBYTE_IC 140000
//SSA 対策用ダミーデータ送信間隔[usec]
#define SLEEP 150000
//一貫性制御間隔
#define REFRESH 100
//最大ロールバック一貫性制御回数
#define RB 5
```



山本 優 (学生会員)

1992年生。2013年福井大学工学部情報・メディア工学科卒業。現在、福井大学大学院工学研究科情報・メディア工学専攻修士課程在学中。



西村 祐介

1987年生。2010年福井大学工学部情報・メディア工学科卒業。2012年福井大学大学院工学研究科情報・メディア工学専攻修士課程修了。現在、トヨタテクニカルディベロップメント株式会社勤務。



福間 慎治

1971年生。1994年長岡技術科学大学工学部電子機器工学課程卒業。1996年同大学大学院工学研究科電子機器工学専攻修士課程修了。1999年同研究科情報・制御工学専攻博士後期課程修了。博士(工学)。同年長岡技術科学大学工学部助手。2000年島根大学総合理工学部助手。2004年福井大学工学部情報・メディア工学科講師。2011年福井大学大学院工学研究科情報・メディア工学専攻准教授。信号処理ならびに信号処理応用システムの研究に従事。電子情報通信学会、計測自動制御学会、IEEE 各会員。



森 眞一郎 (正会員)

1963年生。1987年熊本大学工学部電子工学科卒業。1986年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了。1992年同専攻博士後期課程単位取得退学。同年京都大学工学部情報工学科助手。1995年同助教授。1998年同大学大学院情報学研究科助教授。2006年福井大学大学院工学研究科情報・メディア工学専攻教授。博士(工学)。並列処理、可視化、計算機アーキテクチャの研究に従事。電子情報通信学会、IEEE CS、ACM 各会員。