Optimistic Synchronization of Parallel Simulations in Cloud Computing Environments

Asad Waqar Malik, Alfred Park, Richard M. Fujimoto

グリッドコンピューティング　論文紹介

計算工学専攻　吉瀬研究室
０９Ｍ３８３８７　森　洋介

出典

- Source:
  2009 IEEE International Conference on Cloud Computing
  (September 21-25, 2009, Bangalore, India)

- URL:
  http://doi.ieeecomputersociety.org/10.1109/CLOUD.2009.79

目次

- Abstract
- Introduction
- Issues and Challenges
- Related Work
- A Cloud Architecture for Time Warp
- The TW-SMIP Protocol
- Performance Study
- Conclusions and Future Work

目次

概要

- クラウドコンピューティングは、専門知識や、高性能な計算資源をもたないユーザでも、PDES(並列離散事象シミュレーション)をより広い範囲で利用出来るようになる可能性を持っている
- クラウド内のサービスは、他のアクティブなサービスの共有による負荷で、処理の遅延が起こる可能性があり、それはシミュレーションの実行性能に影響を与える
- 本稿ではTW-SMIPと呼ぶプロトコルを提案し、クラウド環境におけるPDESに関する同期と性能問題に取り組む

目次

背景

- PDES(Parallel Discrete Event Simulation)は色々な様々な分野のアプリケーションに利用され、今後さらに重要性は高まる
  - 大規模の電話通信ネットワーク
  - 製造業
  - 輸送システム
  - 待行列問題

PDES (並列離散事象シミュレーション)

- 内部状態とその状態遷移式を持ったオブジェクト群で構成される世界を考える
- ある初期値から始めて、互いにメッセージを交換しながら状態を更新し、ある時間経過した後の世界を予測を行う
- このモデルは単純な微分方程式では記述できない事象を予測するための手法として、極めて広範囲の応用分野をもっている

PDESの問題(1/2)

- 近年、計算ハードウェアはクロックスピードよりも並列化を意識している
  - Multi-Core Processor
- シミュレータの開発者は、性能向上の手段として、並列性を抽出するという方法をとる必要がある
- PDESの広い普及にはコードを開発・実行できる専門的知識・スキル・実行環境が必要

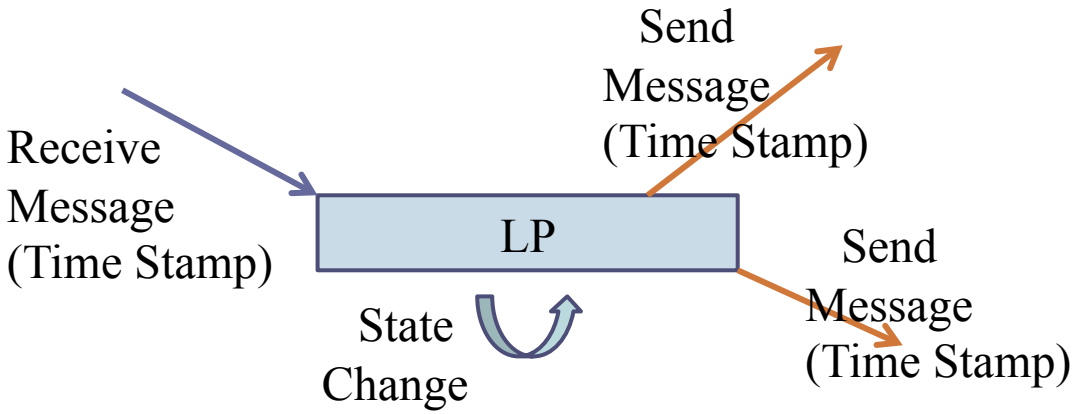
PDESの問題(2/2)

- 広いユーザに利用されるようなシミュレーションシステムに適用されていない
  - 従来のPDES研究の特徴
    - 独自の開発実行環境を構築
    - その上で様々なアプリを開発し、その先進性 や性能や効果を競う
  - 独自環境の上で作られたシステムは、各ノード間の同期と通信プロトコルが独自であるため、 システムの移植が困難
  - キラーアプリに相当するような、広く使われるアプリケーションが生まれていない

参考:ハイエンドコンピューティング技術に関する調査研究
— 高性能プラットフォーム技術を中心とする —
平成13年3月発行 日本情報処理開発協会 先端情報技術研究所

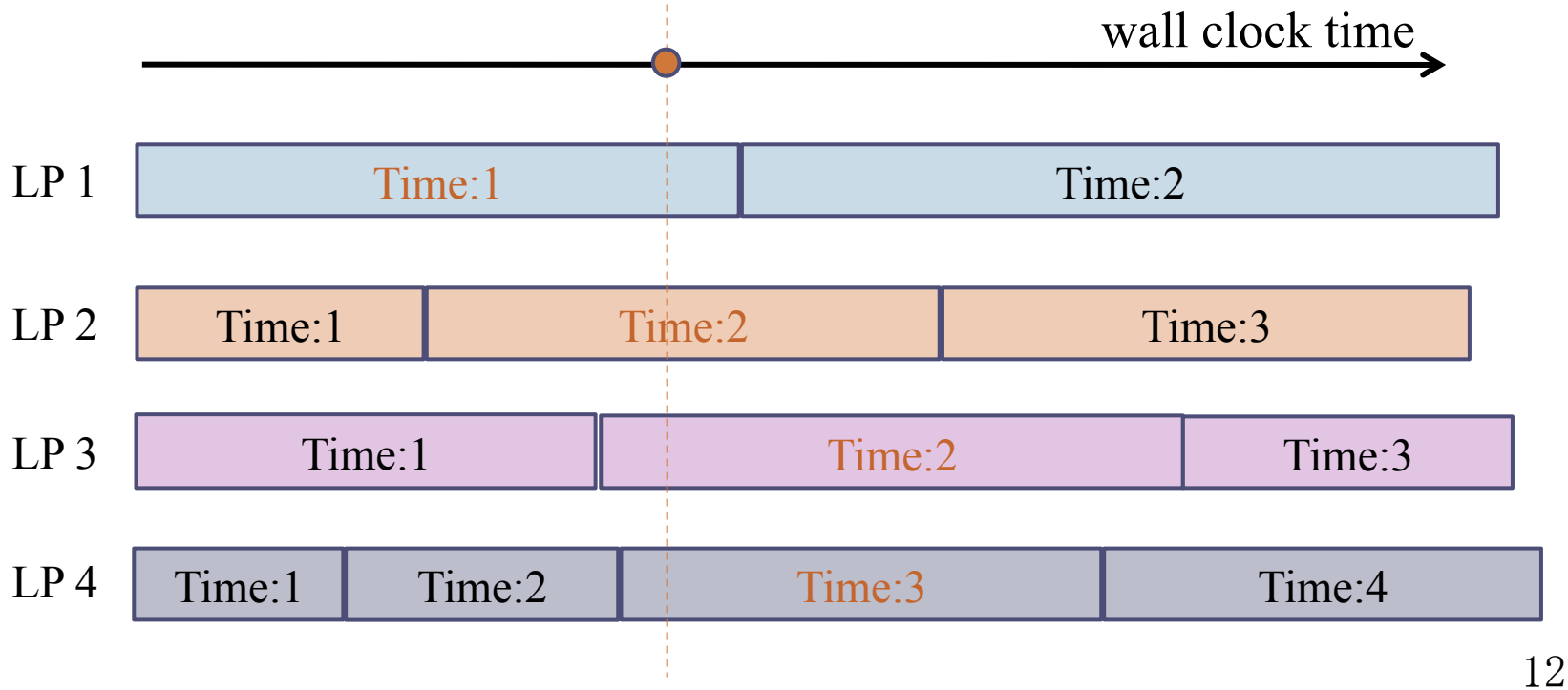
Logical Process

- PDESのプログラムはLP(Logical Process)の集まりから成る
  - PDESは内部状態とその状態遷移式を持ったオブジェクト群で構成される世界
- 各LPはタイムスタンプ（各LPのローカル時刻）を保持し、タイムスタンプと共にメッセージを送る
- 各LPは独自にメッセージを解釈し、状態を更新し、必要ならば他のLPにメッセージを送付するという処理を行う

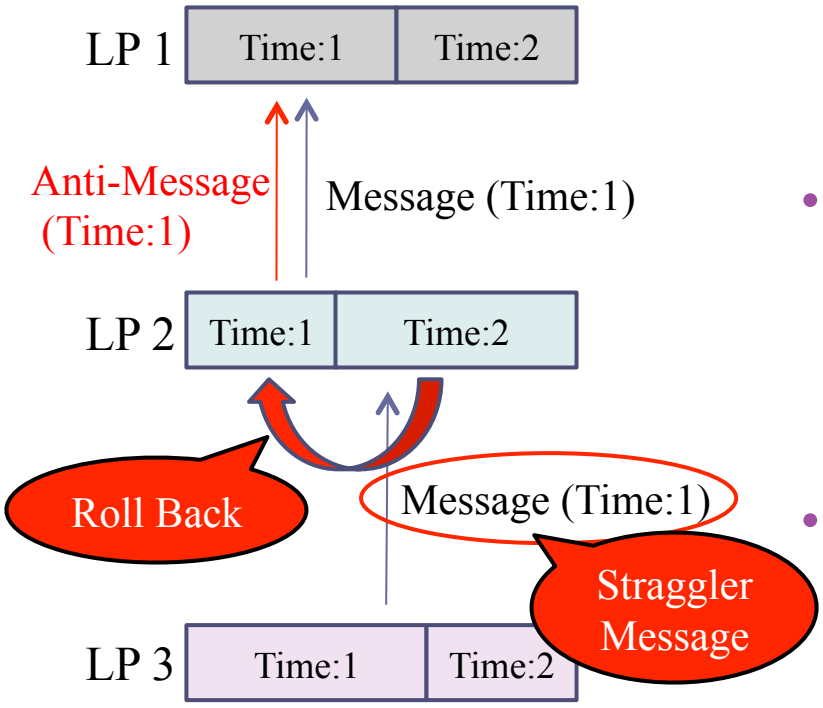


Time Warp (1/2)

- Time Warpは同期問題の取り組みとして有名なアプローチ
- 複数のCPUで構成されたモデルの場合、各LPのローカルな時刻の進行は各々異なる



Time Warp (2/2)



- Straggler Message
  - 各LPが独立した時間管理を持つため、受け取ったメッセージのタイムスタンプが一致しない可能性がある
  - PDESの同期問題
- Roll Back
  - 適切なタイム・スタンプの時刻に処理させるために、進んでしまった 時刻を戻す
  - 各LPは過去の内部状態の変化の履歴を全て保存
- Anti-Message
  - 過去からのメッセージを受け取った瞬間に状態が変わり、 以前に送ったメッセージを取り消す必要がある場合
  - Anti-Messageを送って取り消す

Optimistic Synchronization

- 楽観的（緩い）同期
  - 操作が実際に行われるまでトランザクションの同期を遅延させる。衝突した場合は変更を破棄し、エラーとする
  - Time Warpはこのアプローチ
- 悲観的 同期
  - 最初からトランザクションが並行して実行されるもの見なして同期を行う

Cloud Computing

- クラウドコンピューティング
  - ソフトウェアをサービスで提供、計算資源を仮想化
  - クライアントは離れた場所からアクセスできる
- クラウド環境における想定
  - クラウド上での各ノードは離れた場所に位置する可能性がある
  - 高速ネットワークで互いに接続
  - データとアプリケーションコードは配置・実行ともにクラウド内に存在する
  - 計算ノードは他の計算と資源を共有することを想定

Cloud Computingが提供する可能性

- クラウドコンピューティングはこれらのPDESの問題を解決する可能性を持つ
  - ユーザにPDESな複雑な実行部分を隠すことができる
    - シミュレーションに専門的な知識が必要ない
  - 実行環境を選ばない
    - 広い範囲でシミュレーションを実行できる

TW-SMIP Protocolの提案

- PDESをクラウド上で効率的に実行できるアプローチを提案する
  - 従来のPDESシステムでは、ロールバックが増加してしまう

- TW-SMIP Protocol
  - 局所パラメータとStraggler Messageを元に、動的に各LPの実行を制御
  - バリア同期を避ける
  - 動的にLPの先行実行を制限し、無駄な計算とメッセージを減らす

目次

問題と挑戦(1/2)

- TimeWarpをクラウドコンピューティング上で実行するために必要なもの
  - 資源の効果的な利用
  - 負荷分散
  - ネットワーク衝突と通信
  - 耐故障性
  - プロセス間同期

- これらの機能を自動的かつ、ユーザに中身を意識させずに提供する必要がある

問題と挑戦(2/2)

- 従来のPDESやTimeWarpを利用したアプローチの想定
  - 固定された計算資源
  - 耐故障性を考慮しない
- 本研究での想定
  - 計算資源は複数のアプリケーションが実行され、共有している
  - 実行中に新しいリソース利用できるようになる可能性
  - プロセッサの処理量を計算する際にロールバックされた計算量も考慮する
  - ネットワークの遅延と衝突
  - 耐故障性をもつ
- このような環境で、Time Warpプログラムを効率的に実行するための同期方法を考える

目次

- Abstract
- Introduction
- Issues and Challenges
- Related Work
- A Cloud Architecture for Time Warp
- The TW-SMIP Protocol
- Performance Study
- Conclusions and Future Work

関連研究(1/2)

- LPの先行実行を制限する楽観的制御を採用する同期機構
  - Moving Time Window Protocol
  - Adaptive Time Warp
  - Breathing Time Warp
  - Local Time Warp

- Wolf Calls
  - 本研究に一番近い
  - 連続したロールバックの影響を最小化するために使用する特別なメッセージ

- これらの機構では、クラウド上の取り扱いを考慮していない

関連研究(2/2)

- Volunteer Computing
- Grid Computing

- これらの研究は地理的な分布の問題に取り組んでいる
  - 大きな通信遅延の問題
  - 楽観的同期と資源配分の相互作用の問題に関しての取り組みはない

目次

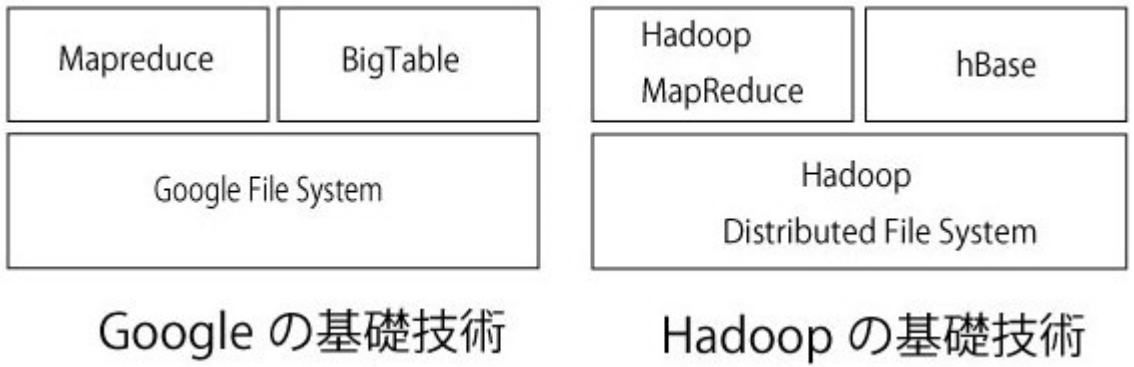
- Abstract
- Introduction
- Issues and Challenges
- Related Work
- A Cloud Architecture for Time Warp
- The TW-SMIP Protocol
- Performance Study
- Conclusions and Future Work

A Cloud Architecture for Time Warp

- 現在の楽観的PDESの機構をクラウド上に実装すると、ロールバックの増加により、パフォーマンスが得られない
- 効率的に実行するには、下層のクラウドインフラを意識した新しいソフトウェアインフラやアルゴリズムが必要
- 既存のプラットフォームであるHadoopが、問題解決の参考になる

Hadoop (1/4)

- 大量のデータを手軽に複数のマシンに分散して処理できるオープンソースのプラットフォーム
- Googleの基盤ソフトウェアであるGFS(Google File System)と, MapReduceのオープンソース実装
- HadoopはHDFS(Hadoop Distributed File System), Hadoop MapReduceから構成されている

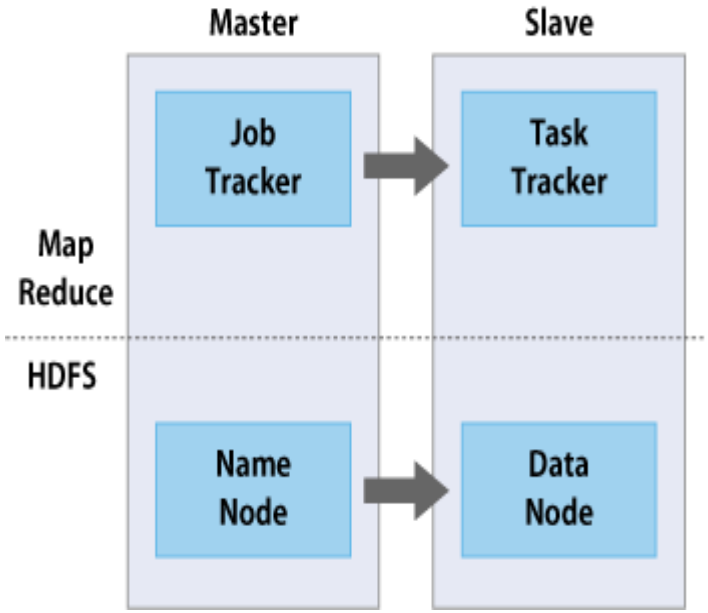


Hadoop(2/4)

- Google File System
  - 大量のデータを安全に保存するための分散ファイルシステム
  - 大容量・スケーラビリティ
  - 耐故障性（同じファイルを複数のマシンに重複して持たせる）

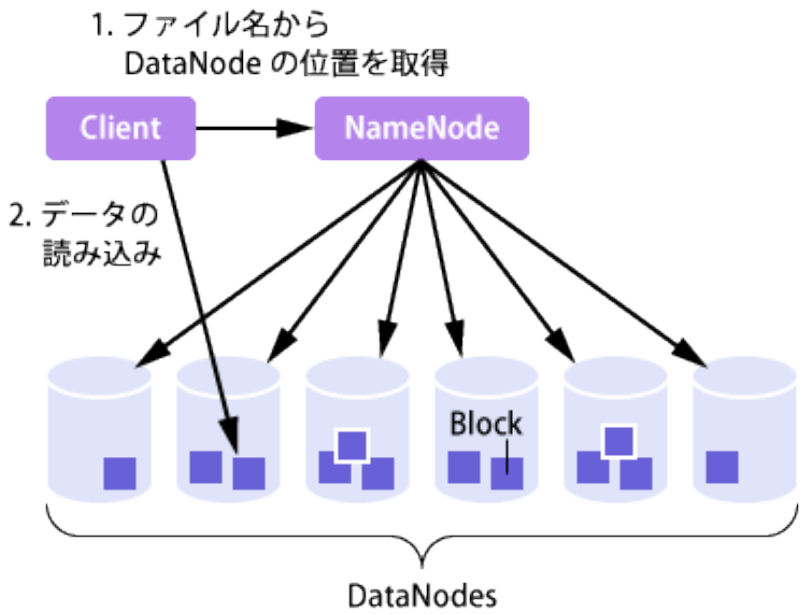
- MapReduce
  - 多数のマシンで効率的にデータ処理を行う目的で考案されたフレームワーク
  1. Map処理: 入力データ（キーと値のペア）を受け取り, 任意の形式に変換し, 必要な情報を抽出する. 全てのMap処理は並列実行可能
  2. シャッフル Mapによって作られたデータを整理し, データを任意の順に並べ替える
  3. Reduce処理 データをまとめて最終的に手に入れたい結果を作り上げ、データ全体についての整理された処理結果を得る

Hadoop (3/4)



- サーバ構成要素
  - Job Tracker(Hadoop MapReduce)
  - Task Tracker(Hadoop MapReduce)
  - Name Node(HDFS)
  - Data Node(HDFS)
- Hadoop MapReduce
  - 「job」はMapReduceプログラムの実行単位
  - MapReduceプログラムを開始すると、クライアントはJob Trackerに「job」をsubmitする
  - JobTrackerは「job」を複数の「task」に分割し、Task Trackerに分配する

Hadoop(4/4)



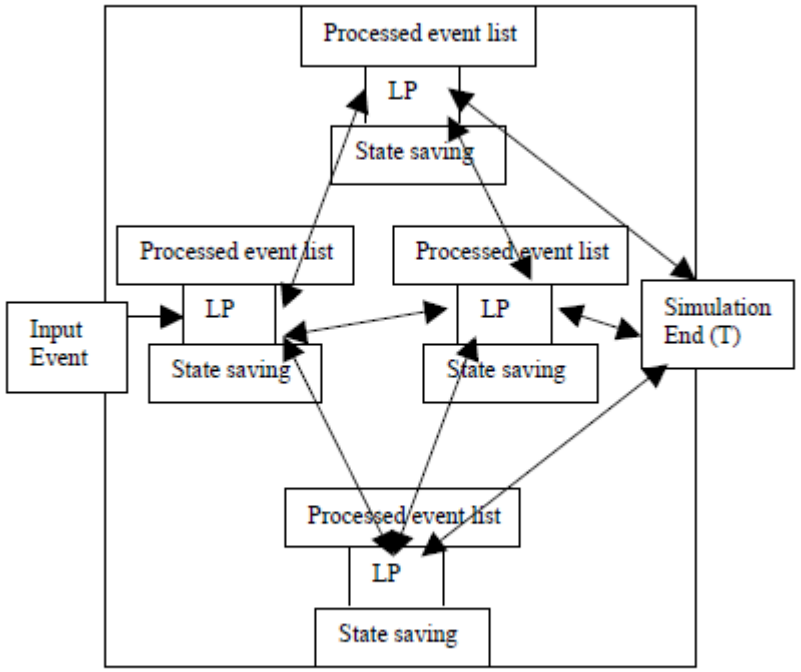
- HDFS
  - Name Node
    - ファイルシステムのメタデータ(ディレクトリ構造・アクセス権)を管理
  - Data Node
    - 実際のデータをブロック単位で保存
- 機能
  - 安全性
    - 複数のDataNodeに同じブロックを保持させる機能
  - 拡張性
    - 動的にDataNodeを追加できる

HadoopをPDESに適用する

- MapReduceのモデルはPDESの実行に利用できる
  - PDESの各LPとメッセージキュー内のイベントのセットはhadoopの「task」に対応する
  - 最初のジョブは「jobtracker」によって登録され、各計算ノードで独立に実行可能な複数のタスクに分割される
  - Map処理はイベントを処理,
  - Reduce処理はこれらのタスクの計算の結果の新しいイベントから分配に使われる
  - HDFSは耐故障性と、各LPのためのイベントキューを提供

- これを元にTime Warp cloud computing architectureを定義

Time Warp cloud computing architecture(1/2)



- 各LPは関係する状態やイベントの情報を持つ
- 各LPへの入力はイベントの集合
- 各LPは並列に実行され、他のLPへ新しい入力イベントを生成する

- Hadoopアーキテクチャを有効に利用
  - 負荷分散
  - 動的な資源配分
  - 耐故障性

- 複数のLPを1つのプロセッサにマップ
  - この手法はhadoopがDataNodeにマッピングする手法と同じ

Time Warp cloud computing architecture(2/2)

- データ構造
  - イベントを保存するイベントキュー
    - 出力キュー
      - アンチメッセージを保存
    - 状態キュー
      - LP状態のスナップショットを保存
    - 処理したもの、処理していないもの両方を保持する
  - ハッシュテーブル
    - イベントキューの中で処理したイベントへの直接アクセス
    - アンチメッセージ/メッセージの組の探索
    - ハッシュ関数は受信したタイムスタンプメッセージと、ソースLP識別子を使ってインデックスを生成
    - 1つのハッシュテーブルは、1つのプロセッサにマップされている全LPのために使われる

目次

The TW-SMIP Protocol(1/2)

- 要求
  - クラウド内で楽観的実行を管理する同期アルゴリズム
  - 高い非同期実行を許す
    - 予測不可能な部分的なLPの停止を許す
    - 過度のロールバックと不安定性に通じる
  - 厳密な同期を要求する機構は望ましくない
  - 疎結合の非同期機構
  - Straggler Messageと過度の楽観的実行による遅延の対処

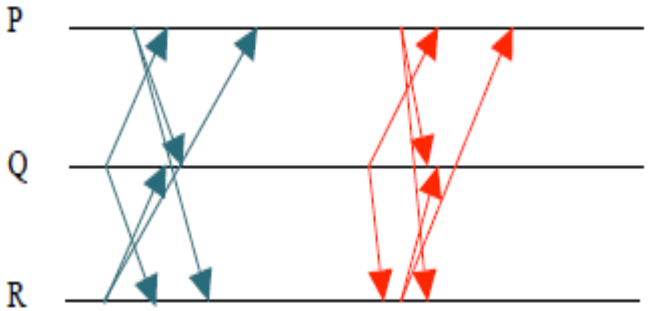
The TW-SMIP Protocol(2/2)

- これらの要求に対し、本稿ではTW-SMIPという楽観的同期のプロトコルを提案
  - そのための定期的な状態メッセージを、HB(Heartbeat)メッセージと名付ける

- Heartbeat(HB) Message
  - Straggler Message検出のために送信メッセージに関する情報を持つ
  - 他のメッセージと比べて高い優先権を与える必要がある
  - 遅延を最小化するためのメッセージバンドリングの対象とするべきではない

HB Messageの生成

- 一定時間後、各プロセッサはstraggler messageの識別フェーズに入る
- 通信している他の全てのプロセッサにHBメッセージを送信
- HBメッセージは、LPの先行実行を制限
- 各LPは他のLPとは独立してHBメッセージを生成



各HBメッセージは同期しない

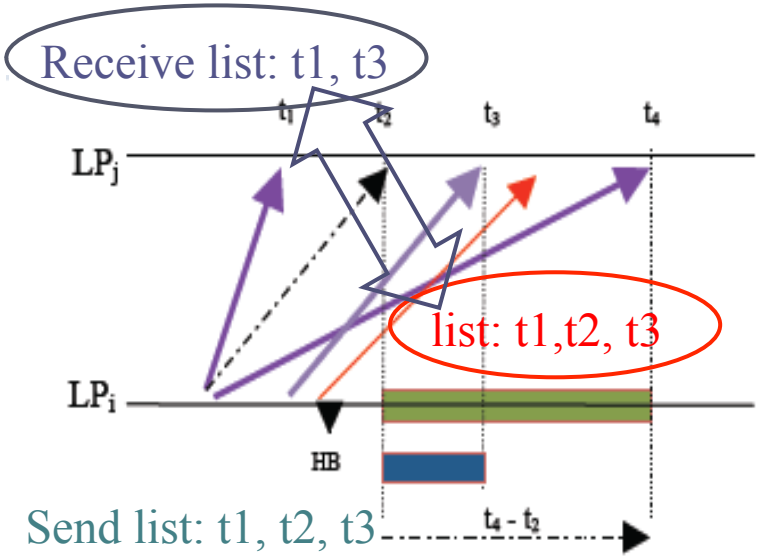
Straggler Messageの検出

- HB Messageは２つの配列フィールドをもつ
  - Time Stamp
  - MID(メッセージ識別番号)
- 配列フィールドは複数のメッセージの情報を保持
- 各LPは、他のLPと送受信した各メッセージのTSとMIDの値を保存
- HBメッセージを受け取ると、各LPはHBメッセージから送信したメッセージの情報を取得
- 各LPは２つのリストを持つ
  - メッセージ到着の際にLPによって保持されたもの
  - HBメッセージの受け取り次第に作られるもの
- この２つのリストが一致しなければstraggler messageが存在する

Straggler Message検出後の処理

- 割り込みを行い、LPはシミュレーションをstraggler messageがあると予測されるポイントにロールバック
- もしstraggler messageのタイムスタンプが送り先のノードのローカル時間より大きければ、LPはstraggler messageが到着した時間とフェーズまでイベント処理を続行し停止する
- HBメッセージの到着が遅れる可能性もある
  - HBメッセージにより識別される全てのメッセージをすでに受信していれば、受信LPはHBメッセージを無視する
  - 加えて受信リストへ登録しているものを除く

Straggler Messageの識別例



1. LPiは3つのメッセージをLPjに送信（タイムスタンプt1,t2,t3)
2. LPiとLPjともに受信メッセージ情報をリストに保持
3. t1とt3のタイムスタンプのメッセージは送信済み、t2は送信中
4. LPiはLPjにHBメッセージを送信
5. HBメッセージを受信するとLPjはHBメッセージとローカルの受信リストの情報を比較する
6. 比較して一致しない場合
   - Straggler messageが存在する
   - t3のイベント処理を止めて、t2までロールバックする、そしてt2を受け取るのを待つ

Three Protocols

- HBメッセージの分配方法によって３種類に区別される
  - Complete network-based SMIP (CNB-SMIP)
    - 各プロセッサは、実経過時間の期間を固定した後にシミュレーションに登場する他のすべてのプロセッサにHBメッセージを送信
  - Partial network-based SMIP (PNB-SMIP)
    - 生成されたHBメッセージは、最後にGVTを計算した後に通信が起こったプロセッサのみに送信する
  - Group-based SMIP (GB-SMIP)
    - この手法は各グループにルートノードが存在するよう、HBメッセージを分配するためにLPを木構造として配置する方法

- これらは、混雑状態、自律管理、通信遅延等のトレードオフにより分けられる

目次

評価環境(1/3)

- 実験環境
  - 均一・不均一な負荷の下でTW-SMIPを評価する
  - 各ノードのスペック
    - Intel Xeon 3.2GHz(dual-core)
    - Memory 6GB
    - GNU/RedHat Linux 64-bit 2.6.9 kernel
  - 各ノードはFast Ethernet(100Mbps)で接続
  - ノード数は14個
  - PNB-SMIPで実装
  - 一つのプロセッサに1000個のLPをマップする

評価環境(2/3)

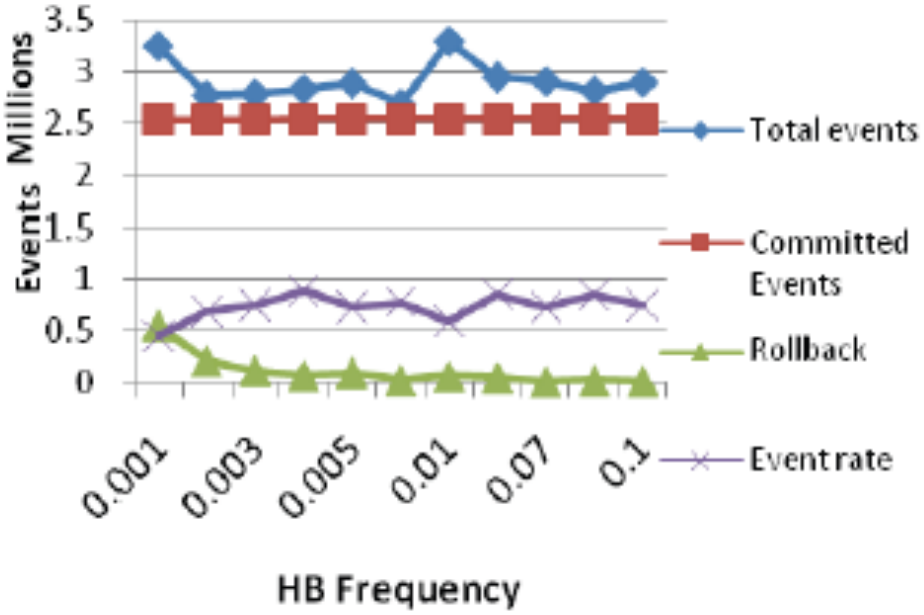
- 評価に用いたテストケース
  - 電力分配システム
- テストケースの挙動
  - 各ノードはソースとして働き、2種類のメッセージを生成する
  - 2種類のメッセージ
    - Self Message
      - ソースノード自身に送るメッセージ
    - Propagating Message
      - ネットワーク内の他のノードに送るメッセージ
  - 各LPは乱数の値に基づき、メッセージを自身か、隣のLPに送るかを決める

評価環境(3/3)

- 不均一な負荷が掛かった状態でTM-SMIPを評価するために、評価はHB頻度と負荷量を変化させて行う
  - Stressというツールを用いてマシンに不均一な負荷を掛ける
- Stress
  - システムのCPU, メモリ, I/O, ディスクへの負荷が設定できる
- Lightly loaded
  - 2つのCPU, 1つのI/O, 1つのメモリアロケータプロセス
- Highly loaded
  - 4つのCPU, 2つのI/O, 1つのメモリアロケータプロセス

性能評価(1/4)

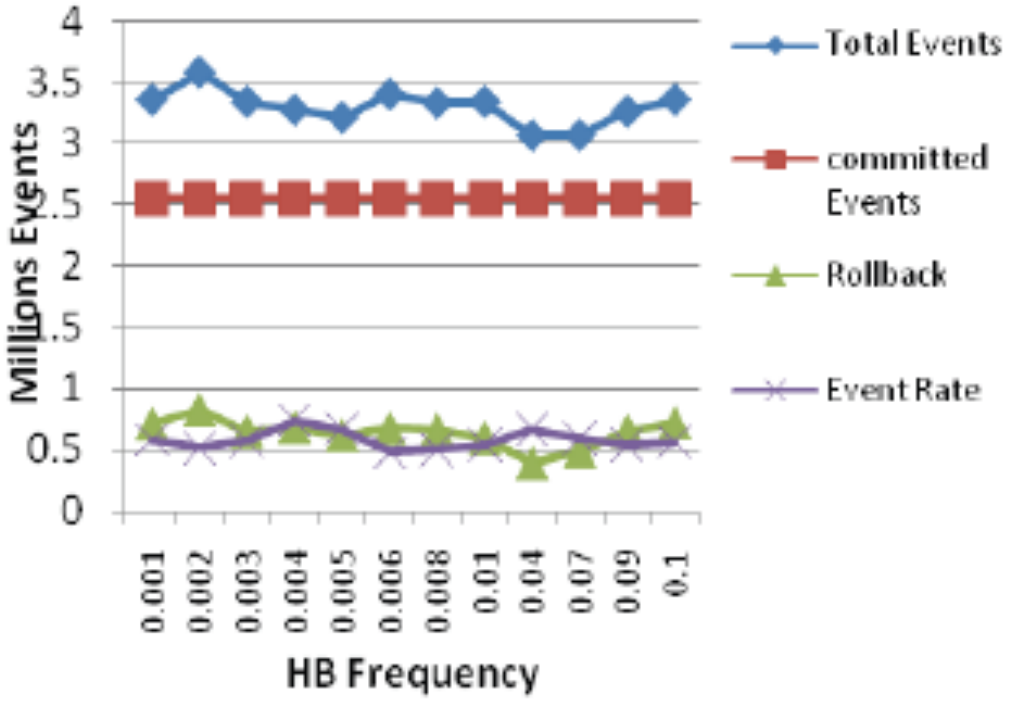
均一の負荷をかけた場合



- HB Frequencyは1秒間にHBメッセージが発生する回数を示す
- HBの頻度を増やすと、rollbackの回数は減少している

性能評価(2/4)

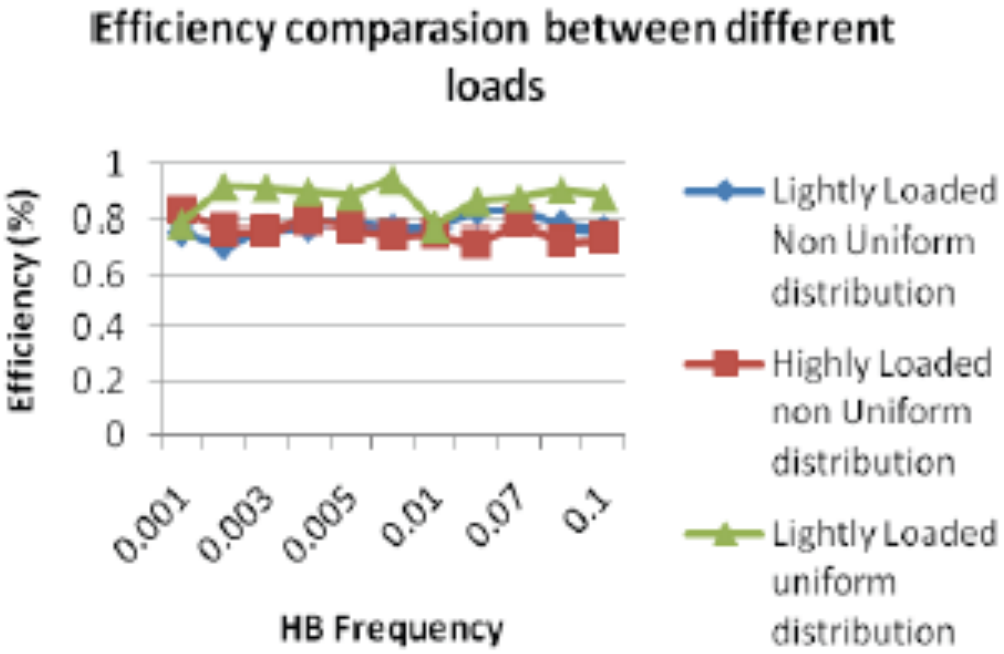
不均一の負荷をかけた場合



- マシンによって負荷の大きさが違う状態
- HBの頻度を変化させても、ロールバックの回数は大きく変化しない

性能評価(3/4)

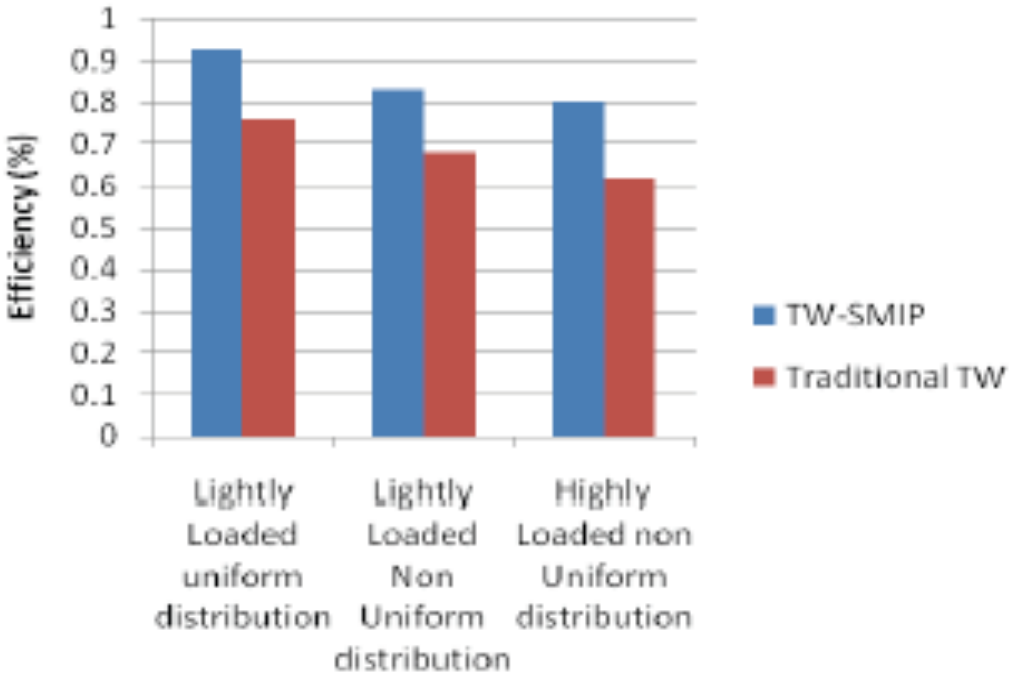
異なる負荷の掛け方の間での性能効率の比較



- ここでの性能効率とは
  処理した全イベントのうち、コミットされたイベントの割合
- 不均衡な負荷分散のケースでは、ロールバックが多く生じ、性能効率が下がる

性能評価(4/4)

TW-SMIPと従来のTime Warp同期機構の比較



- 各テストケースでは、最も高い効率を表したHB頻度を使用
- TW-SMIPは従来のTimeWarpと比較して、大幅な性能効率の向上をもたらしている
  - 従来のTimeWarpでは76%の性能効率
  - TW-SMIPでは91%の性能効率

目次

まとめ

- クラウドコンピューティングは、ユーザにPDESの複雑な部分を見せずに、プラットフォームの実行を保証する
- 従来のTimeWarpフレームワークの下での楽観的なPDESプログラムでは、クラウド上では多くのユーザによる様々な処理とリソースを共有するため、性能がでない
- 新しいプロトコルとしてTW-SMIPを提案する。TW-SMIPはstraggler messageに基づいて、個々のLPに動的に同期ポイントを定義する
- ロールバックするための計算量を減らし、リソースの利用を改善する

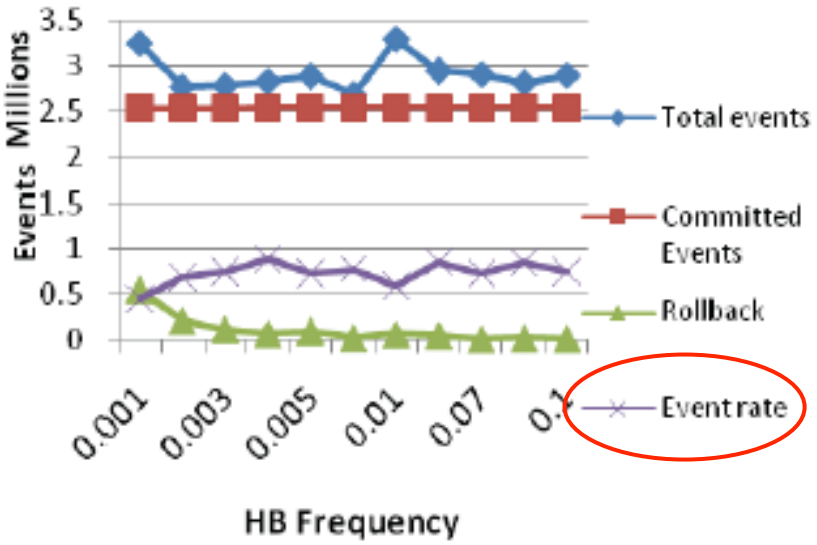
今後の課題

- 本研究は、クラウドコンピューティング上のPDESコード実行における同期問題の初期研究である

- 今後の課題
  - 幅広いPDESのアプリケーションによる評価
  - クラウド環境に向けた、シミュレーションプログラムの実装
  - HB頻度の自動調整機構
  - 相互の耐故障性機構
  - Time Warp内に状態を保存する機構の開発

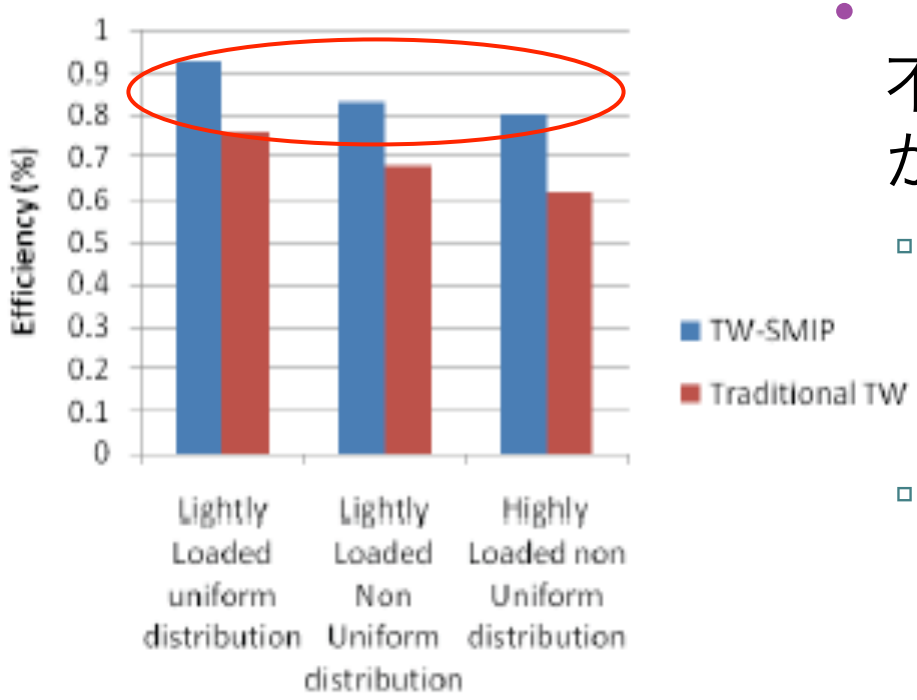
疑問と感想(1/2)

Event rateが何を指しているのか



- コミットされたイベントの割合ではなさそう
  - Efficiencyのグラフと少し異なる

疑問と感想(2/2)



- 「負荷が高い」よりも「負荷が不均衡」であるほうが効率低下が大きい
  - 負荷が小さいノードに、作為的に負荷をかける
  - 負荷分散を均衡にする機構が一番重要