マルチメディア通信と分散処理 77－12
（1996．7．12）

先行制御方式による並列事象型シミュレータについて

鏡味 秀行　渡辺 尚　佐藤文明　水野 忠則

静岡大学 情報学部

並列シミュレーションにおいては、各プロセッサをいかに制御して高速化するかが問題となる。特に待ち行列網シミュレーションのように確率的にイベントを生成するジョブは、前もってスケジュールが立てられないために有効な制御方式に関して十分な知見は得られていない。本研究ではプロセッサの制御方式として先行制御方式を採り入れた並列待ち行列網シミュレータを超並列計算機AP-1000上に実装し、その処理能力について評価を行う。その結果、並列化による効果は得られたもののロールバック等による分散化オーバヘッドがかなり大きい点を明らかにする。そしてロールバックやアンチメッセージの発生頻度を抑えることによって処理能力を向上させる方法をBreathing Time Bucketを基にして構成する。

A Study of Parallel Discrete Event Simulator using Advance Processing

Hideyuki Kagami　Takashi Watanabe　Fumiaki Sato　Tadanori Mizuno

Faculty of Information, Shizuoka University
3-5-1, Johoku, Hamamatsu, 432 Japan

In spite of an expectation that parallel discrete event simulation may provide high performance, it is difficult to control processors to work efficiently. Especially, a simulation model which contains probabilistic events as a queuing network simulation, processes can not be scheduled properly in advance. In this paper, we show an implementation on a parallel processing system, AP-1000 of distributed simulator using Advance Processing scheme similar to Time Warp mechanism to evaluate performance and point out that performance suffers from significant overhead caused by rollback and anti-message. Then we propose an revised Breathing Time Bucket to suppress rollback and anti-messages.

1　はじめに

シミュレーションは、システムを分析、検討する問題解決手法として有効である。近年、計算機技術の発展によりシミュレーションの対象とされる実世界のシステムは大規模、複雑化し、それに対応できるより高速なシミュレーションが求められている。特に計算機のダウンサイジング化が唱えられると共に、従来単一のプロセッサで行っていたシミュレーションを、分割して複数のプロセッサに割り当てることにより高速化を図ろうという、並列シミュレーションが注目されるようになった。

しかし、並列シミュレーションは一般的に確率的不確定要素をふくむジョブであるために、事前に処理計画を立てることが困難で、そのためプロセッサの制御方法に十分な検討が必要である。同じ並列処理の分野でも配列データのベクトル化計算等、単一の処理を異なるデータを相手に行う制御とは明らかに対極をなす。

本研究では、並列シミュレーションの処理能力

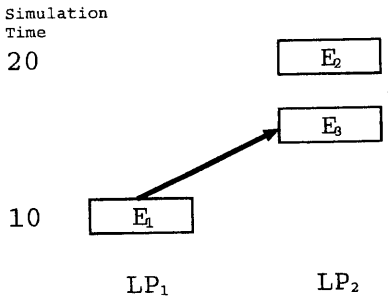

図 1: 従属関係のあるプロセス

向上を目的とし、並列シミュレーションを制御する方式の一つである先行制御アルゴリズムについてシミュレータを実装することにより効力、問題性を検討する。さらに、その問題点を改善する方法の一つとして、改良版 Breathing Time Bucket アルゴリズムを提案しその可能性について検討する。

2 事象型並列シミュレーション

事象型並列シミュレーションは、論理時刻の更新間隔が一定ではないため、プロセッサごとに分割して実行する際にプロセッサ同士の時刻のずれが顕著に現れる。そのためシミュレーションの制御にはさまざまな問題が発生する。

例としてタイムスタンプ 10 の論理プロセス LP_1 のイベント E_1 と、タイムスタンプ 20 の論理プロセス LP_2 のイベント E_2 を考える (図 1)。2 つのイベントが互いに独立しているのであれば E_1 と E_2 は同時実行可能である。さらに、E_2 が E_1 より先に処理されたとしても、なんら問題は起こらない。しかし、もし E_1 を処理することによって、タイムスタンプが 20 より小さい新たなイベント E_3 を LP_2 において生成するのならば、E_3 は E_2 に影響を与えるということから、E_1 が E_2 よりも先に実行される必要がある。もし E_2 が先に処理されると、いずれは生成される E_3 は、過去に影響を与えるという状況を作ってしまう。これは起こってはならない状態であり、nature causality errors(因果律の乱れによるエラー) と呼ばれている。従って、同時実行が可能かどうかを事前に決定できれば良いが、それには E_1 の処理内容を調べる必要があり、一般ににそれをシミュレーションを実行せずに行なうことは困難である。

このように、事象型並列シミュレーションの制御が困難である理由は互いに影響しあう逐次実行の構造が一般的に極めて複雑で、その構造が、内部データに多くを頼っていることである。このような問題に対して過去に解決方法としてさまざまな手法が検討されている。[1]。これらは conservative(保守的) なアプローチと optimistic(楽観的) なアプローチに大別される。

conservative アプローチは、時間的な矛盾の発生を回避することを目的としたものであり、各プロセッサが同期を取りつつ処理を進めて行く。代表的なアプローチとして、Chandy と Misra の Null Message、Peacock の Blocking Table[6] などがありる。conservative アプローチを用いた分散型シミュレータとしては NTT の稲守らによる NEWTS[10]、阪大基礎工の佐竹らによる HASS-QN[12]、慶応大学の相磯らによる KDSS[11] 等が発表されている。NEWTS、HASS-QN は時刻駆動型シミュレータであるがアルゴリズムとしては conservative に分類される。このアプローチの問題点は、時刻の同期を取ることによってプロセッサの遊休時間が長くなり、並列性が十分に生かされないことである。

これに対して optimistic アプローチは時間的な矛盾の発生を許す方式である。この方式では各プロセッサはあらかじめ矛盾の発生に備えて修復過程を用意しておく。各プロセッサは独立に処理を進めて行き、矛盾が起きた際に時刻をロールバックさせることにより整合性を保たせる。代表的なものには、Jefferson や Fujimoto の Time Warp、佐藤、渡辺らの先行制御方式 [3][4] などがある。また、optimistic と conservative を複合させたアプローチもあり、Steinman による Breathing Time Bucket ,Breathing Time Warp [2] などが研究されている。

optimistic アプローチはプロセッサ間で同期を取ることがないため、普段はシミュレーション内で並列性を最大限に生かせるということが利点である。その反面、処理履歴の保存や、時刻矛盾の発見、修復に伴う付加的な作業が必要である。

本稿では先行制御方式、Breathing Time Bucket の 2 つに焦点をあてその性能、コストを検討する。

2.1 先行制御方式のアルゴリズム

ここでは先行制御方式のアルゴリズムの詳細を以下に考察する。

2.1.1 処理時刻矛盾によるロールバック

まず各プロセッサは他のプロセッサと同期を取らず独立して処理を進めて行く。プロセッサ間のメッセージ通信により、論理時刻間に矛盾が生じた場合には、メッセージの受取側のプロセッサが時間を巻き戻す (ロールバック) ことによって矛盾を解消する。ロールバックを行う際、それまで

行っていたイベントは無効となるので、その中にメッセージ送出のイベントが存在した場合には、送信したメッセージを無効にするよう、送り先のプロセッサに要求しなければならない。これをアンチメッセージとよぶ。

2.1.2 履歴保存

ロールバックによる過去のプロセスの状態の再現にそなえ、現在の状況を保存しておく作業が必要となる。イベントを処理する前に全ての状況を保存する方法や、前の状況との差分のみを保存していく方法 [2] などがある。

2.1.3 GVT の検出

ロールバックに備え処理履歴の保存を行っていくと、シミュレーションの進行に伴いそれらの記憶容量は増加する一方である。記憶容量は有限であるので、プロセッサは、これ以前はロールバックが起こらないと保証される時刻を見つけ出し、履歴を除去する必要がある。この時刻はプロセッサ群の論理時刻の最小値であり、GVT(Global Virtual Time) と呼ばれる。GVT を求める手段として、システム全体を管理するマスタプロセッサを設置して、プロセッサ全体に時刻の最小値を要求し GVT をブロードキャストする方法、プロセッサ間に論理リングを作成し、トークンを流し GVT を求める方法などがある。

2.2 Breathing Time Bucket

Breathing Time Bucket(BTB) アルゴリズムはプロセッサ同士の通信は同期を取りつつ処理される一方、各プロセッサの処理は独立して進められるという、conservative と optimistic の両方のアプローチを採り入れている。BTB アルゴリズムは、基本的に、イベント処理、Global Event Horizon の決定、ロールバックとメッセージ伝送、GVT の決定、ガーベージコレクションを 1 つのタイムサイクルとしている。アルゴリズムの詳細を図 2と共に示す。

(1) 各ノードにおいて、プロセッサは同期処理行わずにイベントを処理していく。(図 2のイベント A,B,C) しかし、イベントを処理し終えた後、すぐにはメッセージを伝送しない。その代わり、送信時刻に遅延時間を加えた時刻の中で最小のもの (これを Local Event Horizon(LEH) と呼ぶ。図ではそれぞれ a,b,c の時刻) を、全体のノードの論理時刻を管理するマスタプロセッサ (MP) に知らせる。

(2) MP は各ノードから送られて来た LEH をもとに、最も小さい時刻 (これを Global Event Horizon(GEH) と呼ぶ) をメッセージの配送を許可する上限とし、ノード全体にブロードキャストする。図 2では a の時刻が GEH となる。

(3) GEH より進んでいる時刻のイベントは、同時刻か、またはその時刻より小さい時刻をもつイベントまで、ロールバックされる。ロールバックが終了した時点で各ノードはメッセージを放出する。

(4) GVT を求め、ロールバックが起こらないと保証される時刻に相当するイベントを破棄した後に次のサイクルへと進む。

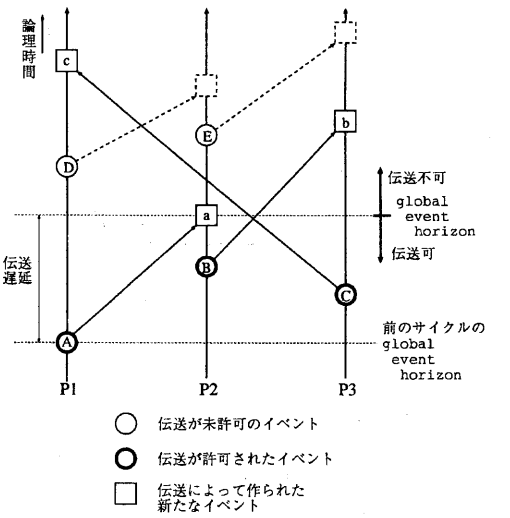

図 2: Breathing Time Bucket のアルゴリズム

このアルゴリズムではアンチメッセージを生成しないように処理を行うため、GEH は LEH の集合の最小値として定められる。その結果シミュレーションの進行は conservative な傾向となる。

3 先行制御方式を用いた待ち行列網シミュレーションの実装

本研究では、先行制御方式による性能特性やコストを検討するために、ノード分散型の並列待ち行列網シミュレータを富士通の超並列計算機 AP-1000 上に実装した。

本研究で使用した静岡大学の AP-1000 は 256 個の SuperSPARC からなり、各プロセッサは 64MB のメインメモリを持つ。プロセッサ間結合網としては 2 次元トーラス状のネットワークを採用して

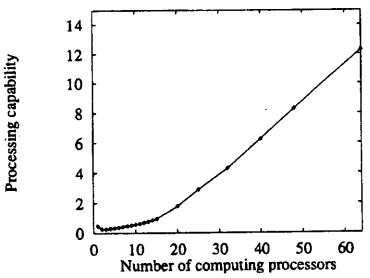

図 3: 先行制御方式のシミュレータの処理能力特性

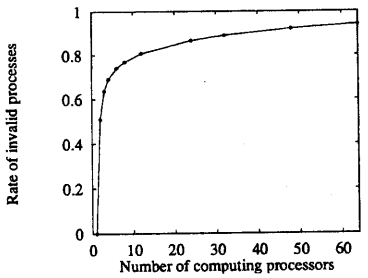

図 4: シミュレーション内での無効処理の割合

いる。また、ワームホールと呼ばれるルーティング方式によりプロセッサ間の通信時間は物理的距離とはほとんど関係なく、隣接していないプロセッサ間でも高速に通信が可能である。

実装したシミュレータはノードの疑似動作、ロールバック処理等を行う複数の Computing Processor(CP) と、GVT を管理する Master Processor(MP)、パラメータの入力や統計の出力を行うホストコンピュータからなる。

対象となる待ち行列網シミュレーションのモデルを以下に示す。

(1) ネットワークはプロセッサ数と同数の 2 ～ 64 の FIFO 型のノードから成る。

(2) 1 プロセッサに 1 ノードを割り当てる。

(3) ノード間は完全網で結合されている。

(4) 各ノードへの到着率は平均 λ(個/sec) の指数分布に従う。

図 3はシングルプロセッサのシミュレータに同等の処理をさせたものを比較対象とし処理能力特性を求めた結果である。ここで

$$\text{処理能力} = \frac{\text{シングルプロセッサでの実処理時間}}{\text{マルチプロセッサでの実処理時間}}$$

である。実験結果から、処理能力はプロセッサ数が 1 の時は 0.47 である。プロセッサ数が 2 台の時には一時下がるが、その後は単調増加となっており、15 台を過ぎたあたりから、シングルプロセッサの性能を上回ることがわかる。プロセッサ数が 1 台の時に処理能力が 1 より少ない理由は、マルチプロセッサシステムではプロセッサが 1 台の場合でも履歴保存を行っており、これによってオーバヘッドが生じているためである。またプロセッサ数が 2 台の場合、メッセージの伝送、ロールバック処理等プロセッサ数 1 台の時に存在しなかった処理によるオーバヘッドのために、処理性能は低下する。シングルプロセッサの場合は、シミュレーション時間はプロセッサ台数 N の増加と共に $o(N^2)$ で増えていく。一方マルチプロセッサの場合の処理時間は、実際には履歴保存のオーバヘッドが存在するが $o(N)$ である。したがって全てのイベント処理を一つのプロセッサで行うシングルプロセッサの方が時間の増分は大きく、結果として性能比は単調増加となると考えられる。

図 4は、シミュレーションの実行時間内で無効な処理を行った時間の割合を示したものである。ここで、

$$\text{無効処理の割合} = 1 - \frac{\text{最適な処理を行った場合の実処理時間}}{\text{シミュレーションの実処理時間}}$$

である。この無効処理時間はロールバックのコスト、アンチメッセージ送信、先行のし過ぎによる無効な処理を全て内包したものである。図より例えば、プロセッサ台数が 40 の場合にはシミュレーション内の無効処理は 9 割である。このことから処理のほとんどは無効処理に費されてしまうことが分かる。無効処理の割合が増加を続けていく理由は、最適な処理にかかる時間の増加率よりもシミュレーション全体にかかる時間の増加率の方が大きいためである。

図 5は、一回のロールバックに対して、アンチメッセージを送る回数、および、そのアンチメッセージがロールバックを引き起こす割合を示したものである。

実験結果から、ロールバックによるアンチメッセージの数はプロセッサ数の増加に対して単調増加の傾向にある。プロセッサ数 40 台の場合、1 回のキャンセルに対して、平均 3 回のアンチメッセージの伝送があり、1 回のキャンセルで平均 0.3 回はロールバックを引き起こすアンチメッセージが発生している。

これらの結果から、プロセッサ台数が増加すれば、シングルプロセッサに比べマルチプロセッサ

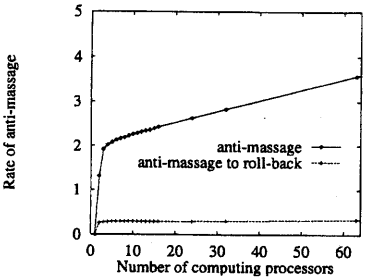

図 5: アンチメッセージおよびアンチメッセージによるロールバックの発生頻度

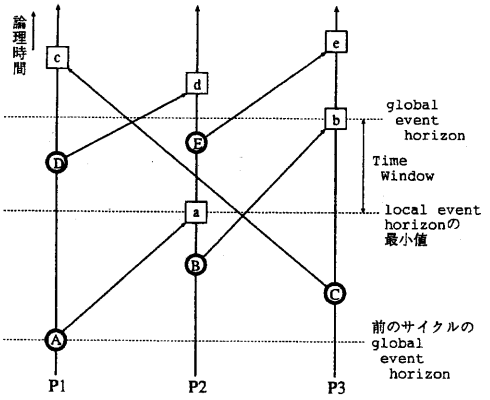

図 6: Time Window を設置した BTB のアルゴリズム

シミュレータの処理能力が向上することが確認できる。しかしその一方、並列化に伴いロールバックやアンチメッセージ等の無効処理に費される割合は高く、処理能力はあまり大きくないと言える。この問題に対する解決策として、

(1) ロールバックにかかるコストを少なくする

(2) ロールバックやアンチメッセージの発生率を抑える。

等が考えられる。(1) については Fujimoto の Direct Cancellation[1] 等が研究されている。また本研究では履歴保存にかかるコストを減らすために枝保存方式 [5] を用いている。枝保存方式ではまず、シミュレーションでイベントを追加していく主となるイベント列を幹と呼ぶ。そして、ある処理を行う前にその後行う予定のイベント列を枝として保存する。ロールバックが生じた場合には矛盾が生じない点まで戻り、そこから出ている枝を新たな幹とする方式である。(2) の解決策については、先行しすぎるプロセッサに歯止めをかける規制先行制御方式 [4]、仮想時刻同調方式 [9]、Brething Time Bucket、optimistic time window[1] 等が研究されている。本研究では主に (2) に着目し、並列性を考慮しつつ、アンチメッセージの発生率を最小限に抑える改良版 BTB アルゴリズムを提案する。

4 BTB アルゴリズムの改良

本稿で提案するアルゴリズムは基本的には Breathing Time Bucket アルゴリズムを踏襲したものであるが、より並列性を意識した方式である。

4.1 改良型 BTB アルゴリズムの詳細

BTB アルゴリズムでは、各ノードがイベントによって生成された新しいイベントのうち、最も小さいタイムスタンプをもつものを LEH とさだめ、その中から GEH を決定している。

これに対して本研究では、LEH の最小値を起点とした time window を設置し、その中でメッセージ群を交換する改良型 BTB 方式を提案する。前のサイクルの GEH と新たな GEH との時間間隔を多く取ることによって同時に処理できるイベント数を増やすことができ、より optimistic な処理を行うことができる。図 6では図 2にくらべ、新たにイベント D,E が伝送可能となっている。

その一方で、time window の範囲内で伝送メッセージを有するイベントが存在した場合はロールバックによるアンチメッセージの発生が生じる。図 6では P2 に a が到着することによって、既に処理を終えている E に時間的矛盾が起き、E は e を無効にするアンチメッセージを伝送する必要がある。しかし、GEH を定めることによって、先行制御方式に起こりがちなアンチメッセージによるロールバックの連鎖反応は削減される。

さらに time window の決定方法によっては、ロールバックの連鎖反応の回数を制限することも可能となる。例えば図 6の LEH 群 a,b,c の中で二番目に小さい値を持つ b の時刻が GEH となるように time window を設定すれば、アンチメッセージによるロールバックの回数を 1 に抑えることができる。

4.2 改良版 BTB アルゴリズムの特性

BTB アルゴリズムが有効に動作する状況は、

(1) 論理時刻上の通信遅延が大きい場合

(2) ローカルで処理されるイベントの割合が多い場合

であり、これは改良版BTBアルゴリズムにおいても同様である。改良版BTBアルゴリズムは、処理の1サイクルを決定するためにイベントによって生成された新しいイベントの時刻を考慮に入れている。したがって遅延が大きい場合、GEHの更新間隔を広く取るような状況を作り出すことができる。同じように、ローカルで処理されるイベントが少ない場合、GEHを短い間隔で頻繁に設定することがない。GEHの更新間隔を大きくするということは、より多くのメッセージを一度に放出することになり、処理効率の向上につながる。

しかしGEHの更新間幅を広く設定するためにtime windowの値を大きく取ることは、ロールバックによって引き起こされるアンチメッセージの数を増加させることにつながる。したがって、time windowの最適値が存在すると考えられる。

5 結論

本稿では、シミュレーションの高速化を図るため、待ち行列網ネットワークをモデルにした並列シミュレーションでより効率よくジョブを処理できるアルゴリズムについて考察した。並列シミュレータはプロセッサごとにジョブを分散させ高速に処理を行うことができる半面、プロセッサ間の論理時刻のずれによる時間的な矛盾に対して付加的な処理を行わなければならない。本稿では、optimisticアプローチの一つである先行制御方式を並列待ち行列網シミュレータに組込み、時刻矛盾がひき起こすロールバック、アンチメッセージなどの無効処理がシミュレーションに与える影響について考察した。その結果、プロセッサ台数が15台を上回った場合に単一プロセッサによるシミュレーションより良いということが明らかになったが、無効処理の割合はシミュレーション全体に対してかなりの割合を示しており、並列化に伴うオーバヘッドが大きいことが確かめられた。

そこで本稿では、ロールバック処理のほとんどをローカルで行い、アンチメッセージの存在を許す代わりに、より並列性を高めた改良版Breathing Time Bucketアルゴリズムを提案した。しかし、並列性を向上させる要因となるGHTの間隔とアンチメッセージの発生率はトレードオフの関係にあり、今後は性能を向上させる最適なtime windowについて評価を行う予定である。

謝辞

本研究を行うにあたって、University of California, Irvineの福田氏からは多大なるご指導をいただきました。ここに感謝の意を表明します。

参考文献

[1] R.M.Fujimoto. Parallel discrete event simulation. *Communication of the ACM*, 33,10(October 1990), pp.30-53.

[2] Jeff S.Steinman. Breathing Time Warp. *7th Workshop on Parallel and Distributed Simulation (PADS '93)*(May 1993), pp.109-118.

[3] 佐藤, 中西, 真田, 手塚　並行処理型待ち行列網シミュレータD-SSQ. 信学論(D)、J69-D,3(1986), pp.302-311.

[4] 渡辺, 中西, 真田, 手塚　規制先行制御方式を用いた非同期ジョブ並行処理システムの処理能力解析. 信学論(D)、J69-D,10(1986), pp.1424-1433

[5] 渡辺　先行制御型マルチプロセッサシステムに関する研究. 大阪大学修士論文 (1984)

[6] J.Peacok,J.Wong and E.Manning. Distributed Simulation using a network of processors. *Computer Networks,3,1*, (1979), pp.44-56.

[7] Reynolds,P.F.,Jr. A spectrum of options for parallel simulation. *Proceedings of 1988 Winter Simulation Conference*(December 1988), pp.325-332.

[8] Livny,M. A study of parallelism in distributed simulation. *Proceeding of the SCS Multiconference on Distributed Simulation 15,2*(January 1985), pp.94-98.

[9] 根本, 高井, 成田　タイムワープ法を用いた離散事象並列シミュレータにおける仮想時刻同調の効果 *Parallel Computing Worikshop(PCW'95 Japan)*(August 1995), pp.P1-D

[10] 稲守, 戸田　複数マスタプロセッサを用いた並列型通信網トラヒックシミュレータの評価. 信学論(B)、J65-B,1(1985), pp.22-29

[11] 中川, 小林, 相磯　データ駆動型離散型シミュレータKDSS-1 信学論(D)、J69-D,3(1986), pp.302-311

[12] 佐竹, 上月, 西田, 宮原, 高島　分散型待行列網シミュレータ. 信学技報, EC83-40(December 1983)