Vol.2010-OS-115 No.17
2010/8/4

HPC 向け VM スケジューラの改良の提案

本庄賢光[†1]　窪田昌史[†1]
北村俊明[†1]

並列処理環境の普及に伴い，MPI(Message Passing Interface) の並列プログラムを，異なる性能のノードから構成される PC クラスタの非均質性とノードの耐故障性に対応させる要求が高まっている．我々は MPI プロセスを Xen の仮想計算機 (VM) 上で稼働させることによって，PC クラスタの非均質性と耐故障性に対応させる手法を提案してきた．しかし VM 上で MPI プロセスを稼働させるとオーバーヘッド大きくなることがある．本研究では，Xen のクレジットスケジューラのタイムスライスとティックの値を変更することと，Domain0 の優先度を変更することによる MPI プロセスの高速化を行った．高速化の評価のためにデュアルコアプロセッサ Core2Duo 3.00GHz，メモリ 4GB のノード上で VM を稼働させ，NAS Parallel Benchmarks 3.3 クラス A のアプリケーション SP の MPI プログラムの実行した．タイムスライスとティックをデフォルトの 30ms，10ms から共に 1ms にし，2 ノード 4 プロセスで実行させたところ 33%高速化された．さらに Domain0 の優先度を 1024 にすることで通常時と比べ，35%高速化された．

Improvement of VM scheduler for HPC

Masamitsu Honjo,[†1] Atsushi Kubota[†1]
and Toshiaki Kitamura[†1]

As HPC and parallel processing environment becomes popular, the demand that parallel programs with MPI(Message Passing Interface) should support heterogeneity on PC clusters with consist of the nodes with different computing power and fault-tolerance of the nodes has arised. We have proposed a technique to support heterogeneity and fault-tolerance for PC clusters by executing MPI processes on virtual machines so that those processes can be easily migrated between nodes. However, when MPI programs are executed on virtual machine Xen, the overhead sometimes becomes large. We therefore make it possible to change the values of the time slice and the tick, which are parameters the Credit Scheduler uses to allocate CPU time, so that the overhead incurred by waiting for receiving data is reduced and the performance of MPI programs is improved. To evaluate the effectiveness of the improvement, we executed the application SP Class A of NAS Parallel Benchmarks 3.3 on virtual machines running on nodes with dual core processor Core2Duo 3.00GHz and 4GB main memory. When SP is executed with four processes on two nodes where the values of the time slice and the tick are both changed to 1ms from the default values of 30ms and 10ms, the performance is improved by 33%. In addition, Domain0's weight is changed from 256 to 1024 ,the performance is improved by 35%.

1. はじめに

近年，パーソナルコンピュータ (PC) の低価格化とパフォーマンスの向上を背景に，多数の計算機をネットワークで接続して構成される PC クラスタがハイパフォーマンスコンピューティング (HPC) の分野などで用いられている．PC クラスタは汎用製品を用いて構成できるため価格性能比に優れ，また計算機ノード数を増減することで利用用途や予算に応じて規模を自由に拡大・縮小できるなどの特徴があり，広く利用されている．

しかし，PC クラスタでは構成要素に汎用部品を多数使用するためシステム全体の信頼性は低い．そのため大規模な計算機クラスタ上で長時間に及ぶ並列プログラムを実行すると，実行中に計算機ノードの故障が発生しプログラムを最後まで実行できなくなることがある．その場合，故障した計算機ノード以外のノードで始めから実行しなおす必要があり，有用な計算が失われてしまう．

そのため，並列プログラミングの MPI(Message Passing Interface) において，大規模な並列プログラムを長時間に渡り実行するためには MPI プロセスのノード間移送を容易にすることが重要となる[7]．

計算機ノードの非均質性と耐故障性の保障，MPI プロセスのノード間移送を可能にする 1 つの手段として，VM(仮想計算機) の利用が考えられる．[1)–4)] しかし，仮想化ソフトウェアで実装されている複数の VM に CPU 時間を割り当てるスケジューラが原因で，VM 上で MPI プログラムを複数のノードで並列で実行した際に，大量のオーバーヘッドが発生している．本研究の目的は，Xen[9)] に実装されているクレジットスケジューラの改良による MPI プログラムの高速化する事である．

†1 広島市立大学
Hiroshima City University

2. 耐故障性と非均質性をサポートする並列プログラミング環境

PC クラスタやグリッドの普及に伴い，アーキテクチャや性能の異なる様々な計算機から構成される並列処理環境が一般的になってきている．また，これらの並列処理環境では，専用並列計算機とは異なり，並列プログラムだけでなく，複数のジョブが CPU 資源を要求して競合しながら処理されることも多いと考えられる．我々は，このような非均質な並列処理環境において，各ノードの性能を引き出しつつ，耐故障性も高めるような柔軟な並列処理環境を提供することを目指している．

2.1 非均質性の対応

並列処理環境の各計算ノードの処理能力に応じて並列タスクである MPI プロセスを分散させるため，抽象プロセッサという概念を導入する．抽象プロセッサは処理能力が全て等しい仮想的なプロセッサである．各抽象プロセッサには MPI プロセスが 1 つずつ対応するものとする．

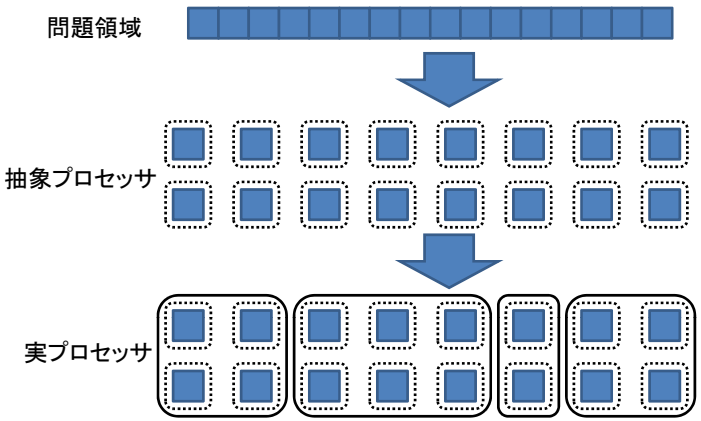

図 1 抽象プロセッサによる問題の非均質分散

計算ノードには，計算ノードの処理能力に応じて複数の抽象プロセッサを割りつけることで負荷の分散を行なう．図 1 に示すように，問題領域 (データや処理) を抽象プロセッサに均等に割り付ける．次に抽象プロセッサを各計算ノードの実プロセッサの処理能力に比例した個数だけ割り付けることで，非均質環境におけるタスク分散を実現している．

抽象プロセッサが並列タスクの最小単位であるため，抽象プロセッサの数を変更することで並列処理の粒度を調整することが出来る．

実行時に負荷が動的に変化するような環境においてプログラムの実行を高速化するため，各ノードに割り付ける抽象プロセッサ数を実行時に変化させて動的負荷分散を行なう．

我々は，計算ノードの処理能力に応じて，抽象プロセッサのノード間移送 (マイグレーション) は，プロセスマイグレーション，あるいは仮想計算機による移送によって実現する．

2.2 抽象プロセッサを用いた耐故障性

PC クラスタ上で MPI の並列プログラムを実行中に，クラスタ内の一部のノードが故障した場合を考える．このとき，故障していないノードを用いて並列プログラムの実行を続けるには，以下の方法が考えられる．

- 待機している代替ノードへ MPI プロセスを割り付け直す (ホットスタンバイ)
- 故障したノードを除外し残りのノードで縮退実行する

故障したノードと同じ性能を持つノードを代替ノードとして使用して並列ジョブを再開させるには，故障したノードで実行されていた抽象プロセッサを代替ノードで実行させればよい．

代替ノードがなく故障していない残ったノードで縮退実行をする場合，残ったノードで負荷が適切に分散されるように抽象プロセッサの再割り付けする．図 2 のように 4 つのノードに多数の仮想プロセッサを割り付けられて並列ジョブの実行中に，1 つのノードが故障して他の 3 つのノードで縮退実行されるとする．16 台の仮想プロセッサを，残りの 3 ノードに，たとえば 7:3:6 の割合で再び割り付ければ，縮退実行時にもノードの性能に応じて負荷を分散することができる．

2.3 Xen

Xen はイギリスのケンブリッジ大学で開発され，2003 年にオープンソースとして公開された．その後は，CPU やサーバを提供している各社からの支援による機能・品質強化が進み，今日では代表的なオープンソースの仮想化ソフトウェアとなった．[5)]

Xen はハイパーバイザ，ホスト，ゲストで構成され, 準仮想化と完全仮想化の両方ともに対応している．Xen の開発は現在も積極的に行われており，2010 年 6 月現在の安定版のバージョンは 3.4.3，耐故障性のための Remus 等の新機能や改良が含まれている最新のバージョン 4.0.0 がリリースされている．

Xen は図 3 のような準仮想化という仮想化手法を採用している．Xen のハイパーバイザ

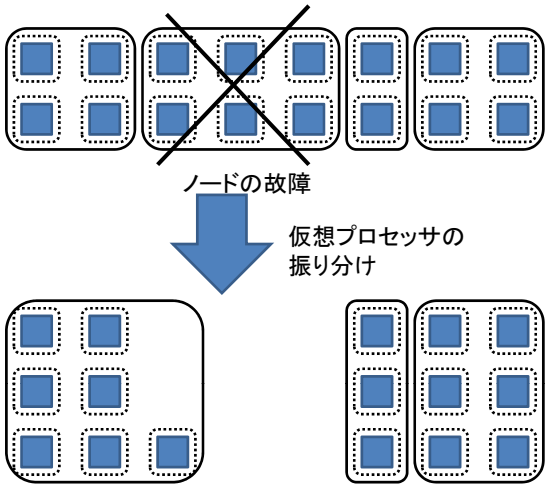

図 2 故障時の縮退実行

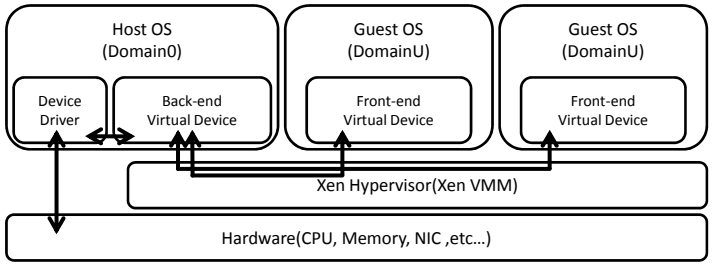

図 3 Xen の準仮想化

自体がハードウェアの管理を行なわず，ホスト OS がデバイスドライバを用いてハードウェアアクセスを行なう．ゲスト OS には仮想デバイスが割り当てられ，仮想デバイスを通じて I/O 要求を行なう．

2.4 クレジットスケジューラ

Xen のハイパーバイザは，稼働中の VM すべてに所定の CPU 時間を与えられるように，CPU 時間割当スケジューラの設計をする必要がある．1 台の実計算機上において複数の VM が稼働している時それぞれの VM すべてに CPU 資源を均等かつ平等に割り当てるために，クレジットスケジューラが用意されている．クレジットスケジューラは，それぞれの VM に weight(優先度) と cap(上限) の 2 つのパラメータが関連付けがされる．weight は VM に割り当てられる CPU 時間の取り分が決定され，デフォルトでは 256 という値が設定されている．

例として，3 つのドメインが次のように設定されているとする．

DomainX weight=256
DomainY weight=512
DomainZ weight=128

どちらのドメインもビジーな状態であると仮定する．ここでは DomainX を基準として，各ドメインの CPU 割当量を考えてみる．DomainY は DomainX に比べ，weight が 2 倍であることから，割り当てられる CPU 時間は DomainX の 2 倍となる．逆に DomainZ は DomainX に比べ，weight が半分であることから，割り当てられる CPU 時間は DomainX の半分となる．cap はドメインに与えられる CPU 時間の上限が決定するパラメータである．デフォルトでは 0 と設定されている．例えば，あるドメインの cap の値が 50 に設定された時，そのドメインに与えられる CPU 時間は通常の 50%となる．

また，weight と cap の 2 つのパラメータの他に，タイムスライス (time slice) とティック (tick) の 2 つのパラメータが存在する．タイムスライスは VM に与えられる CPU 時間 (クレジット) である．実行状態にある VM のクレジットからティック値ごとに差し引いていき，クレジットが 0 になると，実行状態が他の VM に移る．

クレジットスケジューラは稼働している各 VM に weight(優先度) と cap(上限) のパラメータを持たせ，必要に応じて，Domain0 側で各 VM 毎に CPU 時間の割当量を変更する事ができる．タイムスライス，ティックの値はデフォルトでそれぞれ 30ms，10ms とあり，通常は変更する事ができない．そのため，クレジットスケジューラでは，各ドメインに CPU 時間を平等に割り当てるために，ラウンドロビンスケジューリングが採用されている[6]．

2.5 マイグレーション

クラスタ上の並列プログラムの非均質性と耐故障性を提供するにはプロセスのマイグレーションが必要である．非均質性と耐故障性をサポートする一つの手段として，VM のマイグレーションが挙げられる．VM のマイグレーションは，あるホストの上で稼働している VM を別のホストへゲスト OS のイメージを丸ごと移送させることである．Xen では，マイグレーションとライブマイグレーションの 2 種類のマイグレーションを実装している．マイグ

レーションは，VM を 2〜3 秒停止し，停止している間に VM のメモリの内容や OS イメージを他のホストへ移送する．ライブマイグレーションは，VM を一瞬 (1 秒未満) 停止させ，プログラムコードや定数などの不変な部分は一度に転送され，メモリの内容が頻繁に書き換えられている部分の内容は何度も差分転送を行なう．この際の OS イメージは各実計算機が共通で使用している NFS サーバ等から取得するようになっている．

Xen では複数の実計算機間で VM のマイグレーションを際に VM の IP アドレスと MAC アドレスが変化しないため，MPI の場合では，MPI プロセス内でのランク ID も変化しない．そのため，ネットワークコネクションに問題が発生することなく VM や VM 上のプログラムが稼働し続けることが可能である．

2.6 クレジットスケジューラの問題点

MPI プログラムにおいてノンブロッキング通信を行なう上で受信待ちを行なう MPI_Wait や MPI_Waitall という関数がある．VM 上で実行されている 2 つの MPI プロセス間で通信が行われ，一方の MPI プロセスから転送されるべきデータが到着せず，もう一方の MPI プロセスで，これらの通信待ち関数を実行しているとする．受信待ち MPI プロセスが稼働している VM では，受信待ちの間 VM に与えられた CPU 時間を解放しないことによってオーバヘッドが生じている可能性がある．

クレジットスケジューラの内部で設定されているタイムスライスとティックの値がデフォルトでそれぞれ 30ms，10ms に設定されている．上記のオーバーヘッド生じるのは，他ノードと頻繁に通信を行なう MPI プログラムにおいて，タイムスライスの値が 30ms と大きすぎるために他の VM に即座に切り替えることができないからである．

また，Xen は図 3 のとおり，VM(ゲスト OS) の仮想デバイスを通じて他のノード上の VM と通信を行なう．同一ノード上の VM 間で通信を行なう場合はホスト OS(Domain0) の仮想デバイスを経由して通信を行なう．また，ノードをまたがってプロセス間の通信を行なう場合，DomainU の仮想デバイス，Domain0 の仮想デバイス，デバイスドライバ，ハードウェアの順にデータが送信され，受信の場合は，その逆の順番に辿っていく．それらに共通しているのは，Domain0 の仮想デバイスを通じて通信を行なっていることである．各ノード上のプロセス同士で通信を頻繁に行なう場合では，各 Domain(Domain0, DomainU) にデータの転送に必要なタイムスライスが割り当てられていないとオーバーヘッドが発生することがある．

3. クレジットスケジューラへの変更

クレジットスケジューラ (csched_credit.c) はスケジューラの動作や制御についての記述がある．クレジットスケジューラの内部で設定されている定数について説明する．

- CSCHED_TICKS_PER_TSLICE:タイムスライスあたりのティックの回数
- CSCHED_MSECS_PER_TICK:ティックの間隔 (msec)
- CSCHED_MSECS_PER_TSLICE:タイムスライスの秒数 (msec)

csched_credit.c に含まれている定数の CSCHED_MSECS_PER_TSLICE は，通常，CSCHED_MSECS_PER_TICK と CSCHED_TICKS_PER_TSLICE の積となっている (タイムスライスの間隔 = ティックの間隔 × ティックの回数)．2.4 節で述べた weight と cap は Domain0 側のインタフェースを通じて，Xen の管理コマンド xm コマンドで Xen の稼働時に適宜変更することができるようになっている．しかし，タイムスライスやティックについては，通常，Xen の稼働中にユーザが変更することが出来るような機構は備わってはいない．MPI プログラムの種類に応じてタイムスライスやティックの間隔を容易に変更できるように，現在，我々は Xen の稼働中に Domain0 のインタフェースを通じて変更できるように実装を進めている．

また前章で述べたように，Domain0 には適当な量のタイムスライスを割り当てる必要がある．Domain0 などの各 VM のタイムスライスは xm コマンドによって変更可能である．タイムスライスの基準となっている優先度 (weight) を変更することでタイムスライスを 1/2 倍，2 倍，3 倍，…といったように変更することが出来る．

本研究では，Xen の再構築の度に上記の定数の値を変更して，評価を行うこととする．

4. 評　　価

4.1 評 価 手 法

本章では，VM 上で MPI プログラムを実行させる際に，クレジットスケジューラのタイムスライスとティックの値，Domain0 の weight と cap を変化させ，その影響を実測した．評価にはデュアルコアプロセッサを搭載した 2 台の実計算機を使用し，各実計算機には 2 台の VM を稼働させ，図 4 のようにクラスタを構築した．Xen はバージョン 3.4.2 を使用し，実計算機，VM の構成については，表 1 の通りである．実計算機と NFS はギガビットイーサネットで接続され，VM の OS イメージを NFS サーバ経由で取得するようにした．

ベンチマークアプリケーションとして NPB(NAS Parallel Benchmarks)3.3-MPI[8] の

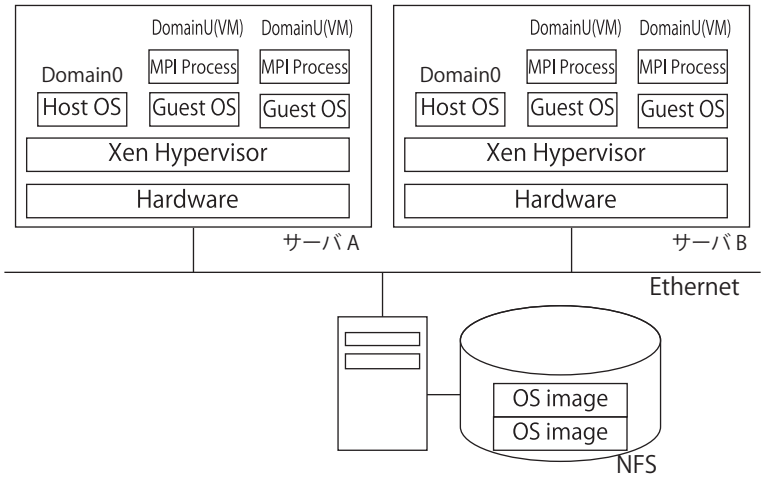

図 4 クラスタの構成

表 1 実行環境

	Domain0	DomainU(VM)	NFS
CPU	Core2 Duo 3.00GHz	1CPU	Pentium4 630 HT 3.00GHz
Memory	4GB	256MB	2GB
HDD	160GB	8GB	1TB
OS	CentOS 5.5 x86_64	CentOS 5.5 x86_64	CentOS 5.5 x86_64
Linux Kernel	2.6.18.8-xen	2.6.18.194.3.1.el5xen	2.6.18.194.3.1.el5

BT，CG，EP，FT，LU，MG，SP を用いた．プログラムサイズはクラス A を用いた．NPB3.3-MPI のソースコードをコンパイルするために PGI Compiler 10.5-0+MPICH1 を使用し，コンパイル時のオプションとして-fastsse を付加し，評価を行なった．

4.2 タイムスライスとティックを変更した場合

クレジットスケジューラのタイムスライスとティックの値を MPI プログラムの実行の度に変更し，2 ノード 4 プロセス実行で評価を行なった．その結果をまとめたものを図 5 に示す．

図 5 は各アプリケーションのタイムスライス (TS)=30ms，ティック (Tick)=10ms の場合の実行時間を基準とし，タイムスライスとティックの値を変更していくことでどれほど

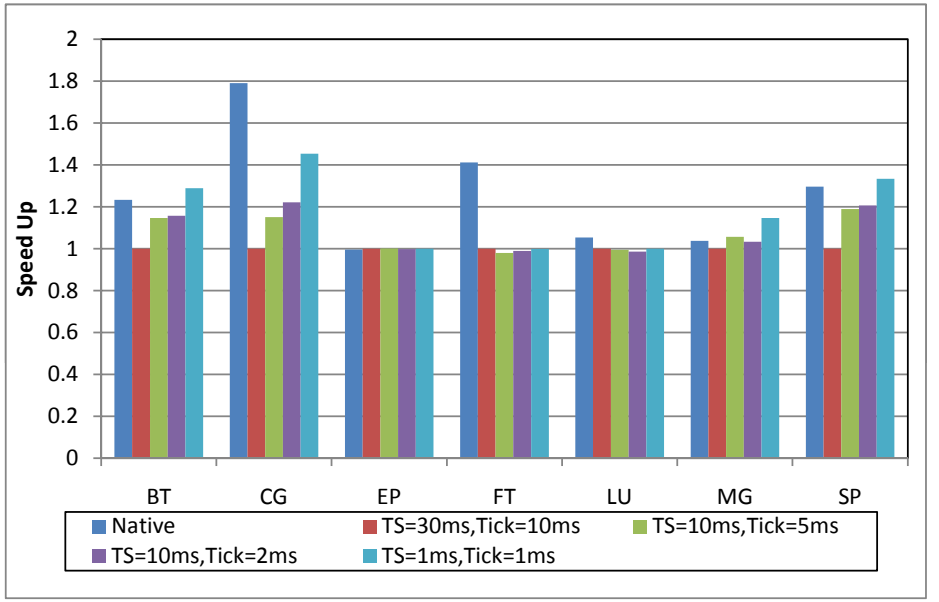

図 5 タイムスライスとティックの両方の値を変更した場合の NPB3.3-MPI の 2 ノード 4 プロセスの実行結果

高速化されたかを示す．ここでの Native は Domain0 の Linux カーネルを通常のカーネル (2.6.18.194.3.1.el5) に変更し，VM(Xen) を稼働させずに，通常の Linux カーネル上で 2 ノード 4 プロセスで実行した時の結果を示す．他はすべて，VM 上で MPI プログラムを実行させた結果を示す．

BT，CG，FT，SP におけるデフォルトの値 (タイムスライス値=30ms，ティック値=10ms) の場合では，Native の実行時間と比較して，VM 上で MPI プログラムを実行させることでオーバーヘッドが発生していることがわかる．BT，CG，SP においてはタイムスライスとティックの両方の値を小さく変更していくことで Native の実行結果に徐々に近づいていき，タイムスライスとティックの値が共に 1ms の場合，タイムスライスとティックがデフォルトの値の時と比較し，BT で約 28.9%，CG で約 45.3%，SP で約 33.4%の高速化がされた．タイムスライスとティックの間隔が 1ms と小さくなったことにより，MPI_Waitall といった他ノードからの通信待ちによるオーバーヘッドが削減し高速化されたものである．

実際に，MPI_Waitall による受信待ちが削減されたかを確認するために，Intel Trace Collector を用いて NPB3.3-MPI SP を 1 ノード 4 プロセス実行時のトレース結果を出力し，

Intel Trace Analyzer で表示させたものを図 6 と図 7 に示す.

図 6 はタイムスライスが 30ms，ティックが 10ms，図 7 はタイムスライスが 10ms，ティックが 2ms の時のトレース結果から，MPI_Waitall を実行している箇所を抜粋したものである．図の青い部分が実際に計算が行なわれている部分で，赤の部分のほとんどが MPI_Waitall である．図 6 の MPI_Waitall の部分が 1.5 から 2.0 秒程要しているのに対して，同様に図 7 では，0.5 秒程度に短縮されていることから，タイムスライスとティックを変更することで，MPI_Waitall 等の他ノードやプロセスからの通信待ちによるオーバーヘッドが削減し高速化されたといえる．

図 5 の BT と SP の実行結果では，タイムスライスとティックの値が共に 1ms の場合，Native の Linux カーネル上での実行結果を上回る結果となった．Native の Linux カーネルのスケジューラを Xen のスケジューラが上回ったということになる．BT や SP といったノード・プロセス間で頻繁に通信を行なう MPI プログラムにおいてはタイムスライスとティックを小さく変更することは有効であると考えられる．

LU ではデフォルトの値のからタイムスライスとティックの両方の値を変更しても，上記のような大幅なオーバーヘッドの削減にはならなかった．

また，EP は両方の値を変更しても Native と特に変化が見られなかった．EP のようにノード間通信が頻繁には行なわれず，CPU 処理が実行の大部分を占めるプログラムにおいては，MPI_Waitall 等の受信待ちが存在しないため，タイムスライスとティックの値を変更することは有効ではないと考えることができる．

このように，タイムスライスとティックの値を小さくすることによって，通信を頻繁に行なう MPI プログラムでは実行が高速化され，通信が頻繁に発生しない MPI プログラムでは実行時間に大きな変化はみられなかった．タイムスライスとティックを変化させることで，VM の受信待ちの際の CPU 資源の消費を削減し，実行の高速化につながったといえる

4.3 Domain0 の優先度 (weight) を変更した場合

VM のタイムスライスの割当量を変更する場合，Domain0 のインタフェースから xm コマンドで各 VM の優先度 (weight) を変更することで，自由に調整することができる．ここでは Domain0 のタイムスライス量を通常の 1/4 から 4 倍の範囲で変化させ，NPB3.3-MPI の各アプリケーションを実行させた．

NPB3.3-MPI の各アプリケーションにおけるタイムスライスやティック，Domain0 の優先度を変化させることによる結果を図 8 から図 14 で示す．デフォルト時 (タイムスライス=30ms，ティック=10ms) の weight=256 時の実行結果を基準とし，各パラメータ毎でどれほどの高速化がされたかを示す．

この時，タイムスライスとティックが共に 1ms，weight が 1024 のとき，基準と比較して CG で 51%，SP で 35%高速化されたといえる．

通信が頻繁には発生しない EP を除き，Domain0 のタイムスライスの間隔が通常の 1/2 や 1/4 といったように十分に与えられない場合，高速化されず，著しい性能低下があった．つまり，ノード・プロセス間通信を頻繁に行なうアプリケーションの実行時には，Domain0 には通常の VM(DomainU) のタイムスライス (優先度) よりも，倍以上のタイムスライスが必要である．Xen の準仮想化の場合，ハードウェア上の NIC や DomainU の仮想デバイスへの I/O 要求に Domain0 の仮想デバイスが応じるためには，Domain0 の仮想デバイスの動作を考慮したタイムスライスを割り当てる必要がある．

4.4 マイグレーションおよびライブマイグレーション結果

図 4 のクラスタの構成に，VM(DomainU) が稼働していない状態の Domain0 を 1 台加える．VM が稼働していない Domain0 はいつでも VM が稼働できるように待機状態となっている．

NPB3.3-MPI SP(クラス A) を 2 ノード 4 プロセスで実行中に，実行開始から約 10 秒後に VM を待機状態の Domain0 にマイグレーション，及び，ライブマイグレーションを行なった結果を図 15 から図 18 に示す．待機状態の Domain0 に VM1 台をマイグレーションすることをマイグレーション 1 回，VM2 台を待機状態の Domain0 へマイグレーションし，もともと 2 台の VM が稼働していた Domain0 が待機状態になるマイグレーションを 2 回とする．

マイグレーション，及び，ライブマイグレーションの結果はタイムスライスとティックの値を小さくすることと，Domain0 の優先度の値を大きくし，Domain0 のタイムスライス量を増やすことでほぼ高速化されている．タイムスライスとティックが共に 1ms，Domain0 の優先度が 1024 の場合，タイムスライス 30ms，ティック 10ms，Domain0 の優先度が 256 のデフォルトの場合で各マイグレーション方法と比較して，NPB3.3-MPI SP(クラス A) の実行時間は，マイグレーション 1 回で 10.2%，ライブマイグレーション 1 回で 9.0%，マイグレーション 2 回で 35.9%，ライブマイグレーション 2 回で 30.7%の高速化がなされた．また，マイグレーション時間は，タイムスライスとティックを変更することによるマイグレーション時間は変化はなかった．通常時の優先度が 256 の場合で 2.6〜3.5 秒要したのに対し，Domain0 の優先度が 64 の場合，3.0〜5.4 秒程度，1024 の場合では 2.6〜3.0 秒程度要した．マイグレーション時はメモリの内容を転送する際に，デバイスドライバを通じてデータ転

送等の I/O 要求をするため，同様に Domain0 には適当なタイムスライスを与える必要がある．

今回は VM(DomainU) のメモリは 256MB であったが，DomainU のメモリがさらに大きく，MPI プロセスがそれに応じたメモリ量を使用することを想定すると，マイグレーション時には Domain0 の優先度によるタイムスライスの変更をすることがとても重要になってくると推測される．

2 章で述べたように，MPI プログラムを実行する PC クラスタを非均質性と耐故障性に対応させるために，MPI プロセスを VM とともにノード間でマイグレーションさせるには，Domain0 の優先度を大きくして適当なタイムスライスを与える必要がある．

5. おわりに

本研究では，HPC 向けの VM スケジューラの必要性とその問題点について考察し，スケジューリングパラメータの変更によるその解決策を提案した．またその解決手法の有効性の評価を PC クラスタ上で行なった．

Xen のクレジットスケジューラのタイムスライスとティックを変更することで，タイムスライスとティックが共に 1ms の場合で NPB3.3-MPI SP で最大 33%，CG で 45%の性能向上を得た．さらに Domain0 の優先度 (weight) を変更することで，Xen のデフォルト状態と比較して NPB3.3-MPI SP で最大 35%高速化された．

Xen のクレジットスケジューラのタイムスライスとティックを変更することが有効であることを示したが，クラウドコンピューティングといった複数の人によって VM が利用される環境下ではサービス品質保証契約 (SLA) で定義されているように，顧客の VM に与える CPU 時間の保証が難しいことと使用状況等の監視ができなくなり，正常な課金ができなくなるといったことが想定されるため，クラウドコンピューティングには適さないと想定される．しかし，HPC 用途といったクローズドな環境下では使用するユーザや用途が限られるため，この場合に限り有効であると考えられる．

参 考 文 献

1) 窪田昌史，前田哲宏，北村俊明, ”非均質環境向けの VM を用いた動的タスク分散システムの構築”, 先進的計算基盤システムシンポジウム SACSIS2007 予稿集, pp.190–191, May 2007.
2) Arun Babu Nagarajan, Frank Mueller, Christian Engelmann and Stephen L. Scott: Proactive Fauult Tolerance for HPC with Xen Virtualization, Proc. Int’l Conf. Supercomputing (ICS), pp.23–32, Jun. 2007.
3) 立薗 真樹, 中田 秀基, 松岡 聡：仮想計算機を用いたグリッド上での MPI 実行環境, 先進的計算基盤システムシンポジウム SACSIS2006 予稿集, pp.525–532, May 2006.
4) Wei Huang, Jiuxing Liu, Bulent Abali and Dhabaleswar K. Panda: A case for high performance computing with virtual machines, Proc. Int’l Conf. Supercomputing (ICS), pp.125–134, Jun. 2006.
5) 小口芳彦, ”仮想マシン道しるべ:基幹サーバへの仮想化ソフト Xen の適用”, 情報処理 Vol.49 No.3, pp.321-324, March 2008.
6) David Chisnall, ”仮想化技術 Xen -概念と内部構造”, (株) 毎日コミュニケーションズ, 2008.
7) 松岡俊輔, 前田哲宏, 窪田昌史, 北村俊明：MPI プログラムの自律チェックポインティング方式の実現，情報処理学会研究報告 2007-HPC-111-11，pp.61-66, Aug. 2007.
8) NASA Advanced Supercomputing (NAS) Division: NAS Parallel Benchmarks, http://www.nas.nasa.gov/Resources/Software/npb.html
9) Citrix Systems, Inc: Xen, http://www.xen.org/

Vol.2010-OS-115 No.17
2010/8/4

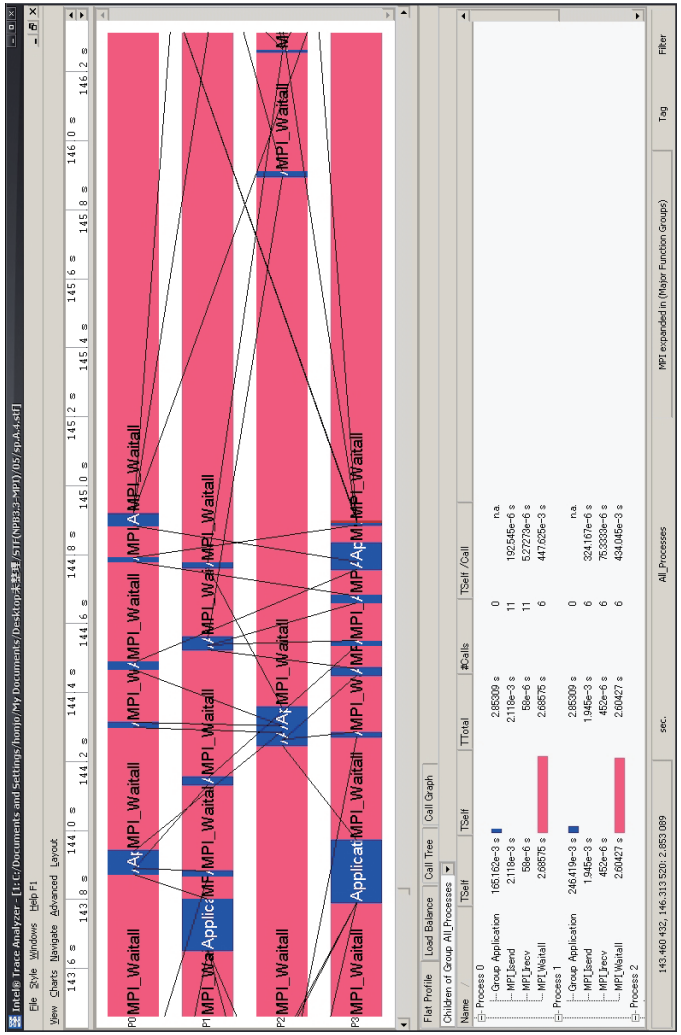

図 6 タイムスライス=30ms，ティック 10ms 時の NPB3.3-MPI SP を 1 ノード 4 プロセス実行時の Intel Trace Analyzer の出力結果

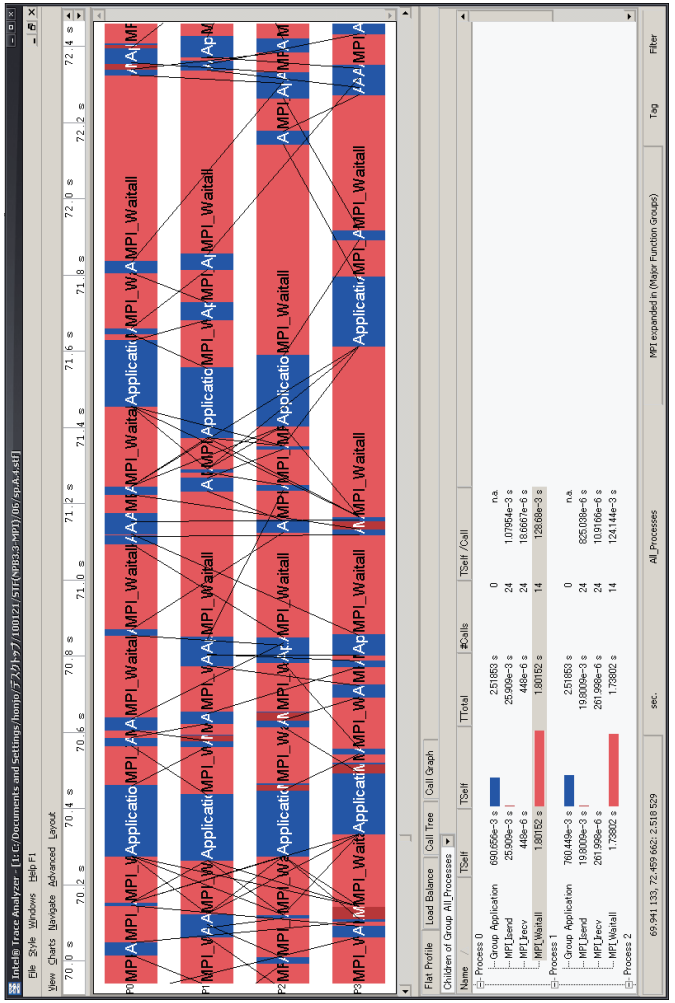

図 7 タイムスライス=10ms，ティック 2ms 時の NPB3.3-MPI SP を 1 ノード 4 プロセス実行時の Intel Trace Analyzer の出力結果

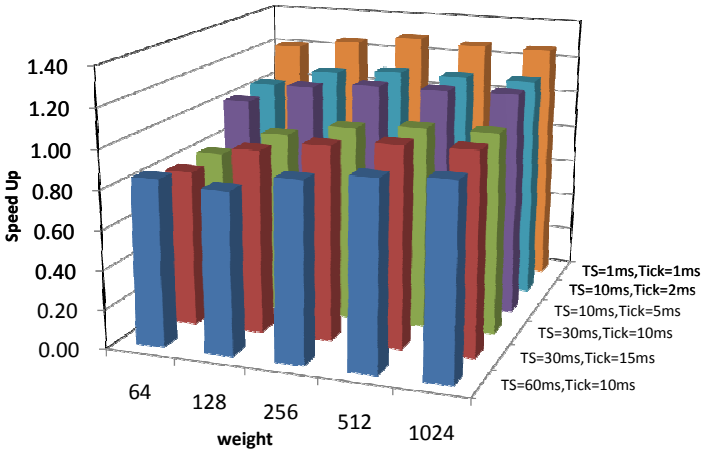

図 8 NPB3.3-MPI BT の 2 ノード 4 プロセスの実行結果

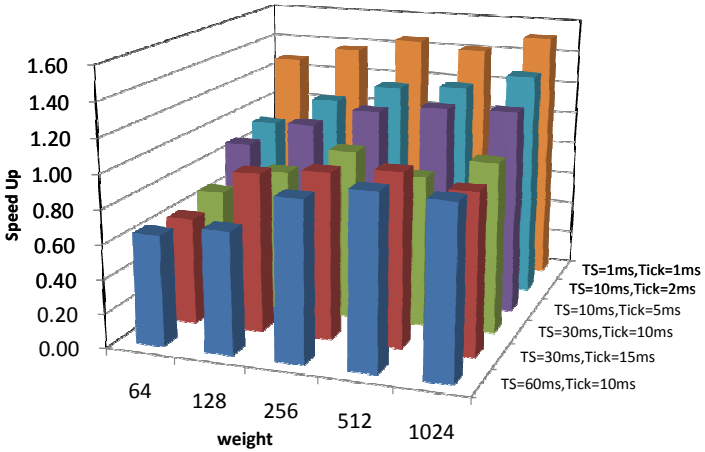

図 9 NPB3.3-MPI CG の 2 ノード 4 プロセスの実行結果

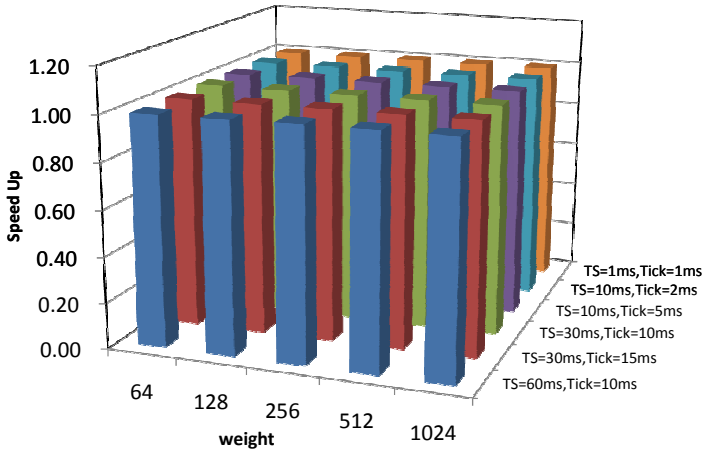

図 10 NPB3.3-MPI EP の 2 ノード 4 プロセスの実行結果

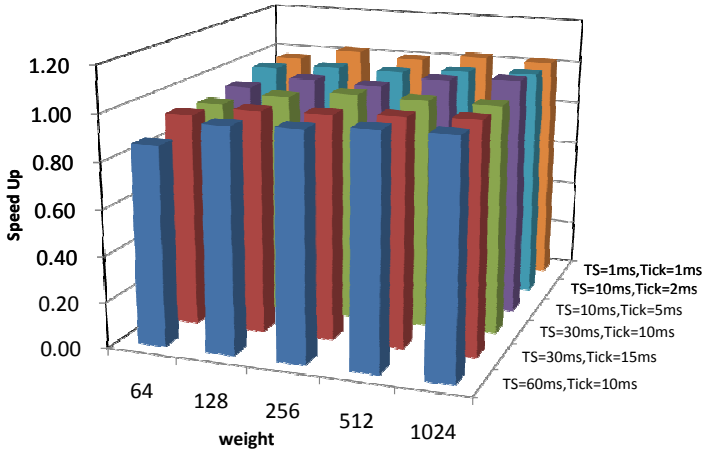

図 11 NPB3.3-MPI FT の 2 ノード 4 プロセスの実行結果

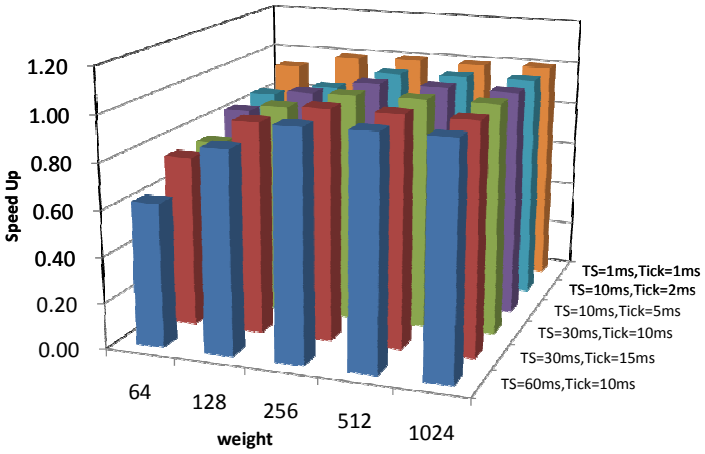

図 12 NPB3.3-MPI LU の 2 ノード 4 プロセスの実行結果

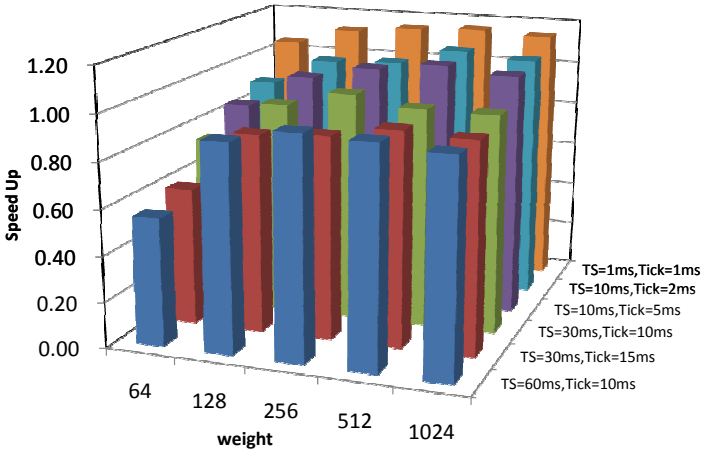

図 13 NPB3.3-MPI MG の 2 ノード 4 プロセスの実行結果

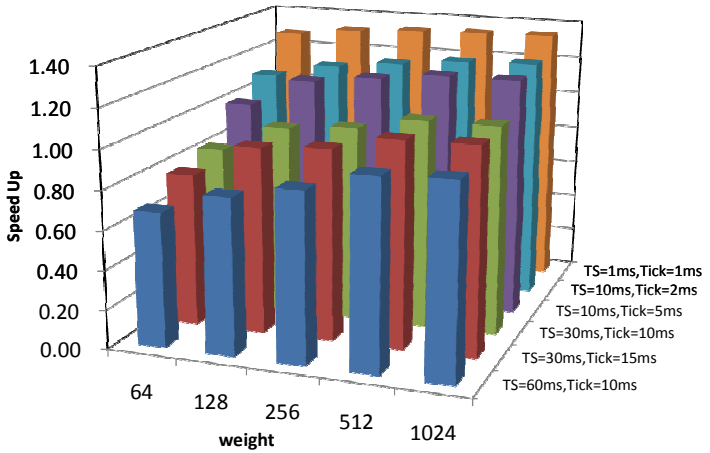

図 14 NPB3.3-MPI SP の 2 ノード 4 プロセスの実行結果

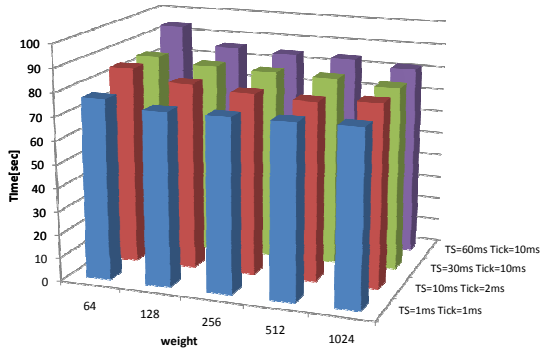

図 15 NPB3.3-MPI SP の 2 ノード 4 プロセスの実行時にマイグレーション 1 回行なった際の実行結果

Vol.2010-OS-115 No.17
2010/8/4

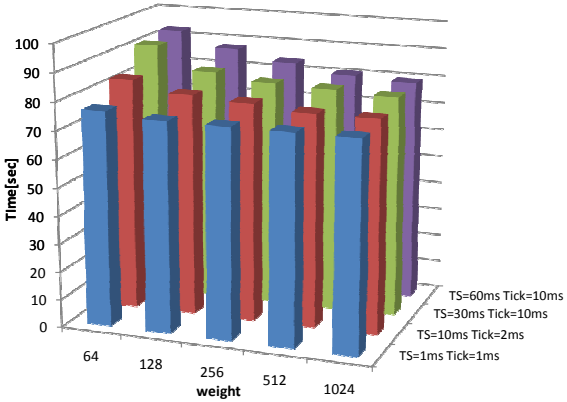

図 16 NPB3.3-MPI SP の 2 ノード 4 プロセスの実行時にライブマイグレーション 1 回行なった際の実行結果

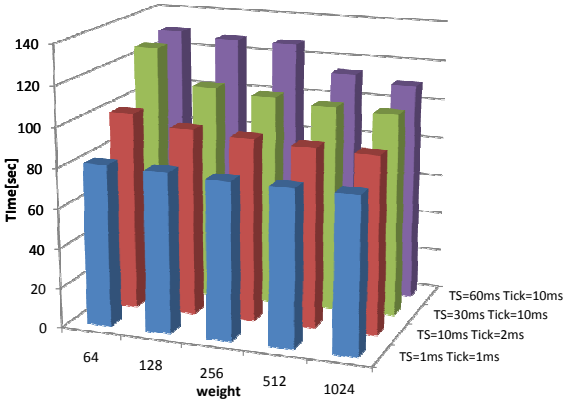

図 17 NPB3.3-MPI SP の 2 ノード 4 プロセスの実行時にマイグレーション 2 回行なった際の実行結果

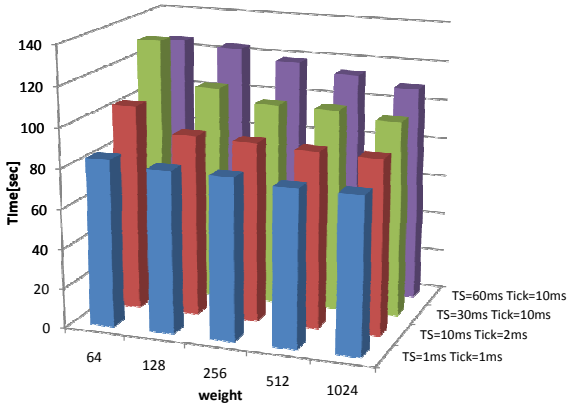

図 18 NPB3.3-MPI SP の 2 ノード 4 プロセスの実行時にライブマイグレーション 2 回行なった際の実行結果