社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－177 (28)
2008－HPC－114 (28)
2008／3／6

グリッド上の計算機エミュレーション手法

西村元一[†1] 合田憲人[†2,†1]

グリッドアプリケーションを開発する上で，アプリケーションの定量的な評価が必要な場合がある．そのために，グリッドエミュレーションシステムが開発されているが，現状のシステムでは計算機負荷の変動を伴うエミュレーションについては十分に議論がされていない．また，計算機台数や計算機性能がエミュレーションに用いる計算機に制限されてしまうといった問題もある．本稿では，グリッド上の計算機エミュレーション手法を提案する．提案システムでは，擬似プロセスを用いて計算機負荷のエミュレーションを行い，仮想計算機と仮想時間を用いて，計算機性能を再現することを可能にする．システムの性能評価を行った結果，ユーザによる効率的なソフトウェアの評価が可能になることが示された．

Emulation scheme of computing resources on the Grid

Motokazu Nishimura[†1] and Kento Aida[†2,†1]

Grid emulation systems, which are evaluation environment for enabling effective grid application, have been developed. In the development of the grid application, a quantitative evaluation of the application has been needed. However, in the existing systems, discussion is not enough about emulating dynamic and time series behavior of computing load. Moreover, systems have a problem that the performance and the number of computers are limited by physical resourceses, which are used for emulation. This paper proposes a computing emulation scheme, which enables to emulate computing load and computing resource by using emulated process and virtual time and machine. The experimental results show the performance of the system to help developers conducting effective evaluation of their application.

1. はじめに

近年，インターネットなどのネットワークに接続された計算機を利用する技術としてグリッドコンピューティングの発展が期待され，様々なアプリケーションの開発が進められている．アプリケーションの有効性を評価するためには，アプリケーションをグリッド上で複数回実行し検証する必要がある．しかし，グリッド上でのアプリケーション開発は，グリッド環境の利用機会の制限や，計算機やネットワークの負荷の変動のため，未だ困難が生じる．

評価環境として，GridSim[1] や Bricks[2] などのグリッドシミュレーション環境に関する研究が進められている．このようなシミュレーション環境では実際の環境をモデル化しアプリケーションの評価を行う．そのため，アルゴリズムの評価には適しているが，アプリケーションを開発するような場合には，アプリケーションを実行可能である環境が必要であり，シミュレーションでは不十分である．

これらの解決方法として，MicroGrid[3] などのグリッドエミュレーション環境が開発されている．MicroGrid ではネットワークの構成にネットワークシミュレータを用いており，ネットワーク構成の詳細な再現が可能である．また，時々刻々と変化するネットワークの状況を設定可能な疑似グリッド実験環境[4] などの開発も行われているが，どの場合も時々刻々と変化する計算機負荷の状況を伴うエミュレーションについては十分に議論がされていない．また，擬似グリッド実験環境においては計算機台数や計算機性能がエミュレーションに用いる計算機に制限されてしまうといった問題点が存在する．

本稿では，グリッドエミュレーション環境におけるこれらの問題点に着目し，これらの問題を解決するためにグリッド上の計算機のエミュレーション手法を提案する．提案手法では，擬似プロセスにより時々刻々と変化する計算機の負荷を再現し，仮想計算機 (VM)

†1 東京工業大学
Tokyo Institute of Technology
†2 国立情報学研究所
National Institute of Informatics

を用いて計算機の障害などの状態を再現する．また，一般にグリッドは性能の異なる計算機が混在するが，これらの計算機の性能差をエミュレーション環境上に再現するために，仮想計算機技術とOS上の時間操作を行うことで，仮想的に計算機性能を変化させ，より柔軟なグリッド実験環境のエミュレーションを可能にした．

以降，2節では，本研究において提案する計算機エミュレーション手法について述べる．その後3節において，提案手法の実装について述べ，4節では，性能評価結果を示す．最後に5節において，本稿のまとめと今後の課題について述べる．

2. グリッド上の計算機エミュレーション手法

本節では，グリッド上の計算機構成や計算機負荷をエミュレーションするための手法について述べる．はじめに，本システムの利用例を以下に二つ挙げる．

- **アプリケーションの耐故障性に関する実験**
  開発したアプリケーションが，計算機の負荷や突然の故障などの障害によって停止することなく，問題なく動作し続けられるかを確認したい．
- **スケジューラの開発**
  グリッド上には多くの資源が存在しており適切に管理することが必要である．開発したスケジューラが意図した通りに動作するのか，様々な計算機構成の元で再現性のある実験を行いたい．

以下では，これらを可能にするための手法について述べる．また，本手法では時々刻々と変化する負荷の再現のために，シナリオシステム[4]を利用する．

2.1 計算機負荷エミュレーション

グリッド上では，計算資源が複数の組織や開発者によって共有されている場合がほとんどであり，他の利用者による計算機負荷は，アプリケーションの挙動に影響を与える可能性が高い．また，これらの負荷は一定ではなく，時々刻々と変化する．開発者は，アプリケーションの開発時にこれらを考慮しながら開発する必要がある．特にスケジューラの開発では，大きな影響を受ける可能性が高い．負荷の変動に影響を受けるアプリケーションの評価のために，グリッドエミュレーション環境にも計算機負荷の変動が再現できなければならない．

本手法では，これを実現するために，擬似プロセスを用いて計算機負荷を再現する．まず，開発者は計算機上で発生させる負荷を表すシナリオファイルを作成する．次に作成したシナリオをシステムに入力すると，管理プロセスが指定した計算機上に擬似プロセスを配置し負荷を発生させる．

2.2 仮想計算機を用いた計算機状態の再現

本節では，仮想計算機を用いて計算機状態を再現する手法について述べる．

グリッド環境に限らず，計算機資源の状態の変化はアプリケーションの動作に影響を与える大きな原因となる．例えば，利用する計算機が予期せぬ障害によって停止してしまう場合も考えられる．

グリッドアプリケーション開発者は，このような障害を考慮し，アプリケーションを開発する必要がある．実環境では，開発者個人の意思によって意図的に計算機障害を起こすことは困難である．本手法では，計算機状態の動的な変化を，仮想計算機を操作することによって再現する．

開発者は，計算機状態の変化をシナリオファイルに記述し，システムに入力する．管理プロセスはシナリオに従って，仮想計算機を操作する．

2.3 仮想時間による計算機間の性能差の再現

グリッド環境では，異なる組織に存在する計算機が利用される場合がほとんどであり，それぞれの計算機性能が全く同じであることはまれである．そのため，グリッドエミュレーション環境上でも計算資源の性能差が再現できなければならない．

本手法では，VM上で動作するOS上の時間を操作することにより，エミュレーション対象計算機間の性能差を再現する．OS上の時間操作とは，エミュレーション対象計算機上の時間 (仮想時間) の進み方をエミュレーションに用いる計算機の時間 (実時間) より遅くすることにより，単位時間内の計算性能を仮想的に変化させるというものである．

本手法では，時間操作のためにGuptaらにより提案されているTime Dilation[5]の方式を利用する．Time DilationではVM上のOSの時間とTimer Interruptの回数を制御することで時間を操作する．また，時間をどれくらい遅らせるかをTime Dilation Factor(TDF)という値を用いて設定する．例えば，$TDF = 2$であれば，実時間上での2秒は仮想時間上での1秒となる．これによって，見かけ上の計算機性能は2倍となる (図1)．

図1 仮想時間による計算機性能の再現

Time Dilation では計算機間に大幅な時間差が生じる可能性があり，グリッド環境を対象とした場合，認証などに問題が生じ，アプリケーションの動作に支障が出てしまう．本実装では，Time Dilation に，以下のような拡張を行い利用する．

- **時間の進み方の精度を上げる**
  TDF によって変更する対象の変数を削減し，時間の進み方に対する TDF の影響範囲を狭めることによって精度を上げる．
- **CPU weight を利用する**
  CPU weight は一定期間内において VM に割り当てる CPU 時間を示しており，CPU 時間を VM 毎に分割し割り当てることで，正確な性能差を再現する．更に TDF と組み合わせることで，より細かな計算機性能差を再現可能にする．

手法では以下の式によって TDF，CPU_{weight} を決める．virtualCPU はエミュレーション対象の計算機の CPU の周波数を示す．realCPU はエミュレーションに用いる計算機の CPU の周波数を示す．NodesOnCPUcore はエミュレーションに用いる計算機上で動作する VM 数を示す．

$$TDF = \left\lceil \frac{virtualCPU[MHz] \times NodesOnCPUcore}{realCPU[MHz]} \right\rceil \quad (1)$$

$$CPUweight = \frac{virtualCPU[MHz]}{\frac{(realCPU[MHz] \times TDF_{max})}{NodesOnCPUcore}} \quad (2)$$

3. 実　　装

グリッド上の計算機エミュレーションを実現するためのシステムの全体設計を図 2 に示す．システムは計算機負荷を生成する擬似プロセスと，計算機状態を再現するための仮想計算機，計算機性能を再現するための仮想時間によって構成されている．

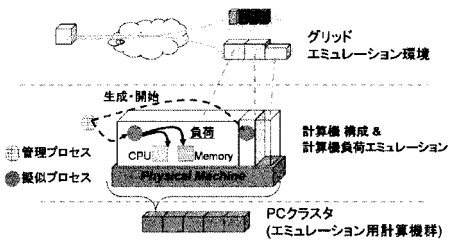

図 2　グリッド上の計算機エミュレーション手法の全体設計

3.1　擬似プロセス

擬似プロセスは計算機シナリオに基づいて計算機負荷を生成するプロセスである．図 3 は擬似プロセスの全体設計である．計算機負荷を生成する負荷生成スレッドとそれをシナリオに従って制御する管理スレッドからなっている．

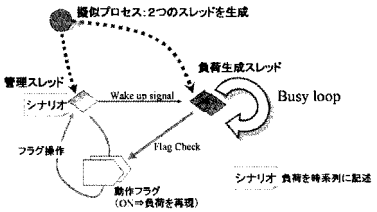

図 3　擬似プロセスの全体設計

3.1.1　負荷生成スレッド

負荷生成スレッドは，生成されてから終了するまで負荷を発生させる．シナリオに従って動作する管理スレッドによって制御される．現在，CPU 負荷とメモリ負荷の 2 種類のスレッドを実装している．

3.1.2　管理スレッド

管理スレッドは擬似プロセス全体の管理を行い，負荷生成スレッドの制御を行う．開始時に計算機シナリオを読み込み，そのシナリオの内容に従って負荷生成スレッドの開始や停止を行う．

3.2　仮想計算機を用いた計算機状態の再現

本節では，仮想計算機を用いてグリッド上の計算機状態を再現するためのシステムの実装について述べる．本研究では，仮想計算機技術として Xen[6] を利用する．

3.2.1　Xen を用いた実装

計算機状態の動的な再現を実現するためのシステムの全体設計を図 4 に示す．Xen の管理ホストである Domain-0 上で，シナリオファイルに従い仮想計算機の状態を操作するためのコマンドを実行し，状態を操作する．

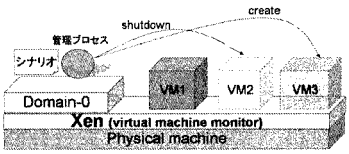

図 4　Xen を用いた計算機状態の再現システム

3.3　仮想時間による計算機性能の再現

本節では，仮想計算機上で動作する OS 上の時間を操作することで，計算機性能を再現するためのシステムの実装について述べる．まず，Xen の時間管理に関して簡単に述べ，その後，本手法の具体的な実装方法について述べる．

3.3.1 Xen での Domain 間での時間管理

Xen では Domain 内の時間は xtime と呼ばれる領域に保存されている．各 Domain の時間は Domain-0 の時間と同期されており，時間の同期は shared_info と呼ばれる共有メモリ領域上の値を用いて行われる．shared_info は Domain-0 のみが変更可能であるため，各 Domain は shadow と呼ばれる領域に値をコピーして利用する．表 1 は shadow 領域内の変数に関する説明である．

表 1 Domain 上の shadow 領域内の変数の一部

変数名	説明
tsc_timestamp	Domain が動作する CPU の TSC
system_timestamp	計算機が起動してからの時間 (秒数)
tsc_scale	一秒間を表す TSC 数

xtime の更新では，各 Domain の開始からの時間が格納されている processed_system_time 変数が利用されており，この値を操作することで，時間を操作することが可能である．processed_system_time は shadow 領域の tsc_scale と system_timestamp を用いて更新される．

3.4 tsc_scale と TDF による時間の操作

提案手法の実装では，時間の操作のために，tsc_scale に注目する．tsc_scale は Domain-0 で定期的に更新され，各 Domain に渡される．図 5 は Domain が tsc_scale を受け取って TDF を反映させ，時間を操作するまでの流れを示している．

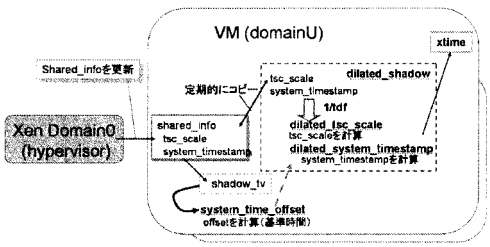

図 5 Domain 内での TDF による時間の操作

Domain が起動すると，まず現在の system_time をオフセットとして system_time_offset に保存する．その後，shared_info の更新の際に得られた tsc_scale と system_timestamp を $\frac{1}{TDF}$ 倍することで，変更された dilated_tsc_scale と dilated_system_timestamp を得る．Domain はこれまでの shadow の値に変えて，dilated_shadow の値を利用し時間を更新する．これによって，Domain 内の時間の進み方は $\frac{1}{TDF}$ 倍になる．

4. 性能評価

本節では，提案システムを実装し，評価を行った結果について述べる．

4.1 擬似プロセスにおける負荷再現性評価

本節では，擬似プロセスでのシナリオの再現性について述べる．プロセスによる正確な負荷を調べるために，プロセスが利用した CPU サイクル数をカウントし CPU 負荷の再現性について検証した．計算機 1 台上でシナリオ変動間隔を 10 秒とし，負荷再現を低負荷 (0〜30%)，中負荷 (30〜60%)，高負荷 (70〜100%) として検証した結果を図 6 に示す．

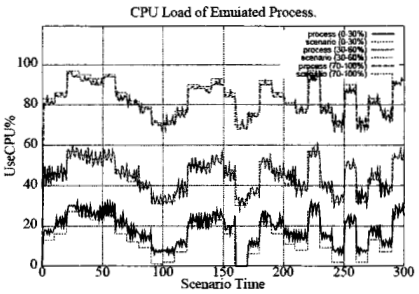

図 6 擬似プロセスにおける負荷再現性評価結果

図 6 より，約 20% 以上の CPU 負荷は正確に再現されていることが分かる．しかし，20% 未満の CPU 負荷については，十分に再現することが出来ていない．これは，低負荷時の CPU 負荷スレッドに対する管理プロセスの切替が十分でないことと，OS 上でのプロセススケジューリングによるものと考えられる．

4.2 計算機状態の再現性評価

この実験では，1 台の仮想計算機に対して計算機状態の再現性評価を行った．なお，計算機の状態は稼動中と停止の 2 通りとし，ping の返答が返ってきた場合は稼動中と判断した．本実験に用いたシナリオは，開始から 300 秒後に停止，600 秒後に稼動，900 秒後に再起動，1200 秒後に停止するといったものである．検証した結果を図 7 に示す．

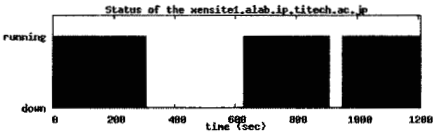

図 7 計算機状態の再現性評価結果

結果をシナリオに沿って見ていくと，300 秒付近で計算機が停止しており，シナリオ通りに状態が変化していることがわかる．同様に 600 秒付近では計算機が起動して稼動状態になっていることが分かり，システムが正しく動作していることが分かる．

4.3 仮想時間による計算機性能の再現性評価

実験では，TDF を 1〜10 と変化させることで，計算機性能の再現が正確に行われるかを検証した．モンテカルロ法を用いて π を求めるプログラムを用いて，点の数を 10^8 に固定し，その実行時間を計算機上で計測した．プログラムは 5 回実行し，結果にはその平均を用いている．図 8 に各 TDF における実行時間の結果と，$TDF = 1$ の実行時間を基準としたときの速度向上比を示す．結果から，エミュレーション対象計算機上での実行時間が $\frac{1}{TDF}$ になることが確認できる．

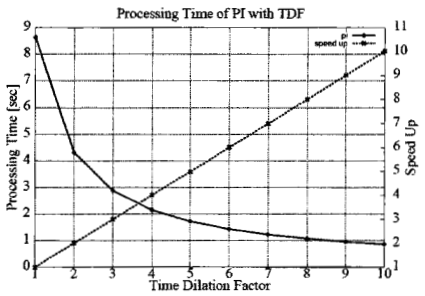

図 8 TDF の変化による実行時間と速度向上比

次に，TDF と CPU_{weight} の変動による計算機性能の再現性について評価した．これにより，CPU_{weight} を変更することで計算機性能を分割可能であることを示す．TDF を固定し CPU_{weight} を 100%，90%，80% ... 10% と変化させる．図 9 は $TDF = 1, 2, 5, 10$ とした場合の実験結果である．図では CPU_{weight} が 100% の時の実行時間を基準に理論値を設定し，計測結果と比較した．それぞれの TDF において，理論値とほぼ等しい結果を示していることがわかる．よって，CPU_{weight} と TDF によって計算機性能を設定可能である．

4.4 グリッドエミュレーション環境の評価

本節では，提案手法を用いてグリッドエミュレーション環境を構築し，その環境上でのグリッドアプリケーション実験評価について述べる．

4.4.1 対象とする実グリッド実験環境

本実験で対象とするグリッド環境の概要を図 10 に示す．

対象としたグリッド環境は，2 つのクラスタを持つ

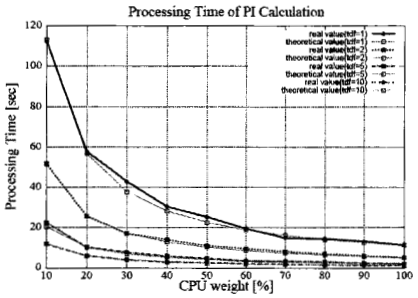

図 9 CPU_{weight} の変動による計算機性能の再現性

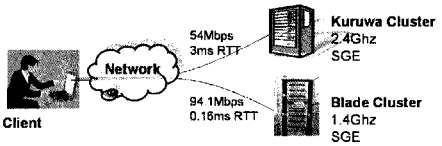

図 10 対象とする実グリッド実験環境

実グリッド環境である．図中のネットワーク性能値に関する測定値はそれぞれ，遅延の計測には ping を利用し，帯域の計測には Iperf[7] を利用した．

4.4.2 グリッドエミュレーション環境

本実験では，クライアント計算機 1 台，クラスタのエミュレーションに用いる計算機 2 台，ネットワークエミュレーションに用いる計算機 1 台からなるグリッドエミュレーション環境を用意した．表 2 は実験環境上での各仮想計算機の TDF と CPU_{weight} を示している．

環境を設定した後，ネットワーク性能について計測した結果，client-Kuruwa 間でネットワーク帯域 54.7Mbps，RTT3.03ms，client-Blade 間でネットワーク帯域 95Mbps，RTT0.732ms を得た．実環境上の値と近いため，ネットワークの再現は正常に動作しているといえる．

4.4.3 対象とするグリッドアプリケーション

利用したアプリケーションは，Ninf-G[8] のサンプルコードである pi_wait_any を用いた．概要を図 11 に示す．このアプリケーションでは，クライアント側でリモート関数コール (GRPC call) が呼び出されると，リモートで π の計算を行い，その結果をクライアント側へ返す．GRPC call の対象となる計算機が複数存在する場合，計算が終わった方の計算機に対して次の GRPC call を実行する．

4.4.4 グリッドアプリケーションの実行結果

実験では，5 回の RPC call を行い，プログラムの実行開始から終了までの時間を計測し評価した．それ

表 2　グリッドエミュレーション環境の計算機構成

計算機名 (クラスタ)	TDF	CPU	CPU_{weight}	Memory	役割
kuruwa-gw (Kuruwa)	6	2.4GHz	0.20	256MB	ゲートウェイノード
kuruwa01,...,05 (Kuruwa)	6	2.4GHz	0.20	256MB	計算ノード
gk (Blade)	3	1.0GHz	0.08	256MB	ゲートウェイノード
blade01,...,05 (Blade)	4	1.4GHz	0.12	256MB	計算ノード

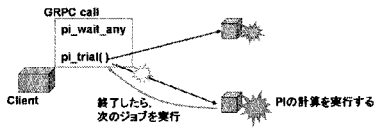

図 11　対象とするグリッドアプリケーション pi_wait_any

ぞれのクラスタ毎に実験を行い，実行時間を計測した．図 12 は両クラスタでの実行時間の比較である．

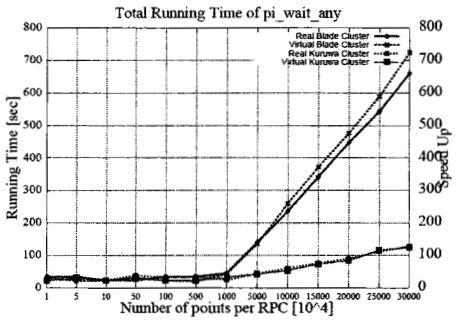

図 12　Blade, Kuruwa Cluster における実行時間の比較

両クラスタ共に，実環境に近い実行時間を得られている．よって，グリッド上の計算機性能差を再現できているといえる．

5. まとめと今後の課題

本稿では，グリッド上の計算機エミュレーション手法を提案し，その実装について述べた．計算機エミュレーションでは，CPU やメモリの計算機負荷を再現するための擬似プロセスを提案し，実験によって負荷を再現可能であることを示した．また，計算機状態を再現するために仮想計算機を利用する方法や，仮想計算機上の OS 上の時間を操作することで，計算機性能差を再現する手法について述べ，手法を用いてグリッドエミュレーション環境を構築し，アプリケーションを用いて検証を行ったところ，計算機性能差が期待通り再現されることが確認できた．これにより，グリッドアプリケーション開発者は，グリッド上の計算機を考慮したアプリケーションの開発が可能となる．

現状のシステムでは，計算機性能の設定値 (TDF や CPU_{weight}) の決定方法が完全ではなく，これによって性能差の再現が正しく行われないことがある．よって，更なる検討が必要である．今後は，大規模なグリッドエミュレーション環境の構築や，時間操作のための API の開発を検討中である．

謝辞　本研究の一部は，科学研究費補助金基盤研究 (B)(課題番号 18300018) による．

参 考 文 献

1) Murshed, M., Buyya, R. and Abramson, D.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, pp.1175–1220 (2002).
2) Takefusa, A., Matsuoka, S., Nakada, H., Aida, K. and Nagashima, U.: Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms, *In Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pp.97–104 (1999).
3) Liu, X., Xia, H. and Chien, A.: Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics, *Journal of Grid Computing*, Vol.2, No.2, pp.141–161 (2004).
4) 笠井武史，西村元一，前田高宏憲人：グリッドエミュレーション手法に関する研究，*IC2006*, pp. 175–180 (2006).
5) Gupta, D., Yocum, K., McNett, M., Snoeren, A.C., Vahdat, A. and M, G.: To Infinity and Beyond: Time-Warped Network Emulation, *3rd Symposium on Networked Systems Design and Implementation (NSDI)* (2006).
6) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *ACM Symposium on Operating Systems Principles (SOSP)* (2003).
7) Iperf: Iperf - The TCP/UDP Bandwidth Measurement Tool. (`dast.nlanr.net/Projects/Iperf`).
8) Takemiya, H., Shudo, K., Tanaka, Y. and Sekiguchi, S.: Development of Grid Applications on Standard Grid Middleware (2003).