2003－EVA－6 (4)
2003／5／30

可用性を考慮した グリッドコンピューティングの性能評価

国本　賢一
株式会社　アイ・アイ・エム　技術部　プロダクトセンター

小型コンピュータやワークステーションの性能の向上、ネットワークの普及に伴い、分散処理システムのあり方が変りつつある。大規模タスクを分散処理するためのシステムも同様で、特にグリッドコンピューティングにみられるような参加、不参加が比較的自由な分散処理システムの応用が多くみられるようになった。それらの技術の普及に伴って、グリッドに特化した性能の評価手法に対する注目が高まってきている。本研究は、マルコフ過程を応用した並列コンピューティングの評価モデルに対し、対象とするグリッド環境に適合させるためにいくつかの修正を行う。また、その結果モデルに与える影響を、シミュレータによる実験結果と比較することで検証し、その有効性を探る。

Performance Evaluation in Consideration of Availability of Grid Computing

Kenichi Kunimoto
Product Center, Engineering Department, IIM Corporation

The state of a distributed processing system is changing with improvement in the performance of small computers or workstations, and the spread of networks. Similarly, the system for carrying out the distributed processing of the large-scale tasks, and many application of a distributed processing system with comparatively free participation and nonparticipation which are seen by grid computing came to be seen. The attention to the evaluation technique of a performance in which it specialized in the grid has been increasing with the spread of those technology. To the valuation modeling adapting the Markov process of parallel computing, in order to fit this research to the target grid environment, we make some corrections. Moreover, this research verifies by comparing the influence which it has on a model as a result with the experiment result which used the simulator, and explores the validities.

1. 導入

グリッドコンピューティングの普及の背景には、小型コンピュータやワークステーションの性能の向上、ネットワークの普及などがあげられる。我々は、安価なコンピュータをネットワークにつなげることで、コンピュータ資源を他人に貸したり、また、利用することができるようになってきている。様々な応用技術が一般化するにつれて利用者が増え、取り扱うジョブも大きくなり、そしてグリッドにつながれるコンピュータの数が増大していく。そういった中で、特に運用計画という面から、グリッドに対する性能評価、予測の需要が高まってくることは必然である。

大規模なジョブを複数の並列に実行可能なタスクに分散して処理するためのグリッド環境について性能評価を行うとき必要になる指標には、タスクの数、利用可能なプロセッサの数、可用性、タスクやジョブの実行時間、合計稼働時間などがあげられる。フォールトトレランスや性能が一定ではない分散処理システムの性能評価に関する研究[2]が古くからあり多くの成果を上げている。しかし、プロセッサが頻繁に入れ替わるようなグリッド環境に見合った評価手法については、まだ多くの研究余地が残されている。本研究は、マルコフ過程を応用した並列コンピューティングの評価モデル[1]に対し、対象とするグリッド環境に適合させるためにいくつかの修正を行う。また、その結果モデルに与える影響を、シミュレータによる実験との比較により検証し、その有効性を探る。

本研究は、ある一つの大きなジョブを複数の並列実行可能なタスクに分散して、独立した複数のプロセッサで分散処理するグリッドを対象とする。問題を簡単にするため、モデルの構築にあたってネットワークや集中処理の負荷が十分に分散されているものと仮定し、その影響がごく微量であるか、無視できるものとする。またタスクの数とプロセッサの台数は共に有限とし、システムに含まれる稼働中と非稼働中のプロセッサの最大数を P台、与えられるタスクの数を N個とする。また、システムに与えられるタスクの数がプロ

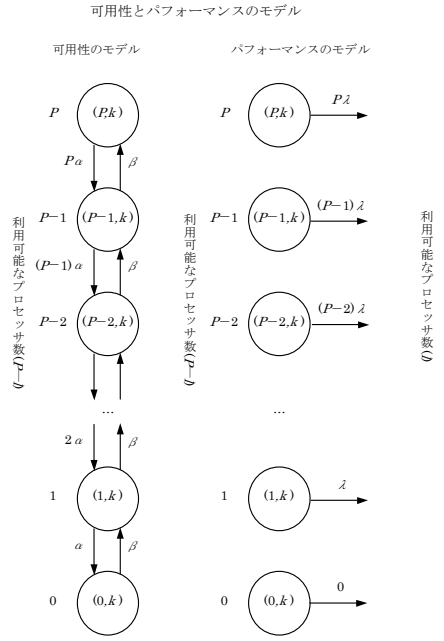

図１： 可用性のモデル（左）とパフォーマンスのモデル（右）

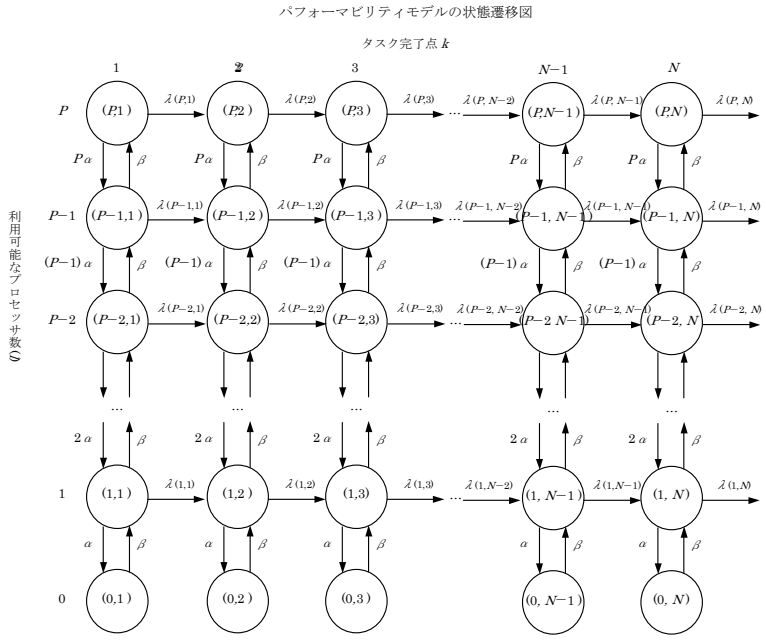

図 2： 可用性とパフォーマンスのモデルからパフォーマビリティモデルを構築する

セッサの台数に対して十分に大きいもの（$N >> P$）と仮定する。

次章では、グリッドを評価するためのパフォーマビリティモデルと、その派生モデルについて解説する。第３章では、パフォーマビリティモデルとその派生モデルによる性能評価を行い、その有効性をシミュレーションを使用して検証する。

2. 評価手法

グリッドの性能を評価する指標には、ジョブの実行時間、各タスクが完了するまでの時間、ジョブ実行中のプロセッサの稼働率、ワークなどがある。これらは、システムを構成するタスクの数、プロセッサの数、可用性などをパラメータとするモデルから求めることができる。この章では、評価に使用されるパフォーマビリティモデルと、その派生モデルについて解説する。

2.1. パフォーマビリティモデル

本研究は、可用性がグリッドの性能にどのような影響を与えるかを調べるための、マルコフ過程応用した並列分散処理システムのパフォーマビリティモデルを使用する。このモデルは、システムの性能を表すパフォーマンスのモデルと可用性を表すモデルの二つを組み合わせて構築されている。

パフォーマンスのモデルを図１の右側に示す。右向き矢印の上の式は、並列実行中のタスクのうち、最初の一つが完了するまでの処理比率を表している。このモデルではタスクの処理時間は、処理比率をパラメータにとる指数分布に従うランダムであると仮定している。プロセッサが一台稼働中の場合の処理比率を λ とすると、P 台稼働している場合のスループットは $P\lambda$、$P-1$ 台稼働している場合は $(P-1)\lambda$ となる。この図は、稼働中のプロセッサの台数に比例して、タスクの処理比率が高くなり、処理時間が短くなることを示している。

次に、信頼性のモデルをあらわす遷移図を図１の左側に示す。グリッドのプロセッサはジョブの実行中にオフラインになったり、逆にオフラインだったプロセッサがオンラインになったりする場合がある。これを、マルコフ過程《Markov Process》[3]の一つである機械修理工のモデル《Machine Repairman Model》の故障と修復による状態遷移に当てはめ、図にしたのが図１の左側のモデルである。このモデルでは、並列に稼働中のプロセッサのうちどれか一つが故障するまでの時間が、故障の比率をパラメータにとる指数分布に従うものと仮定している。稼働中のプロセッサの数に比例して故障の比率は高くなり、故障するまでの間隔は短くなる。一方、機械修理工のモデルでは故障したプロセッサは、一定の比率をパラメータにとる指数分布に従う時間で修復されることを示している。この図から、あるジョブを稼働中のプロセッサの台数は、故障と修復の比率によって変動する。プロセッサがタスクの実行中に故障した場合、そのタスクは失敗したものとみなされ、別の待機中のプロセッサが見つかるまで保留される。新しいプロセッサに失敗したタスクが割り当てられると、それはもう一度はじめから再実行される。プロセッサが修復された場合、または、タスクの処理が完了した場合、プロセッサは待機状態になる。このとき、割り当てられていないタスクがあれば、即座にプロセッサに割り当てられる。

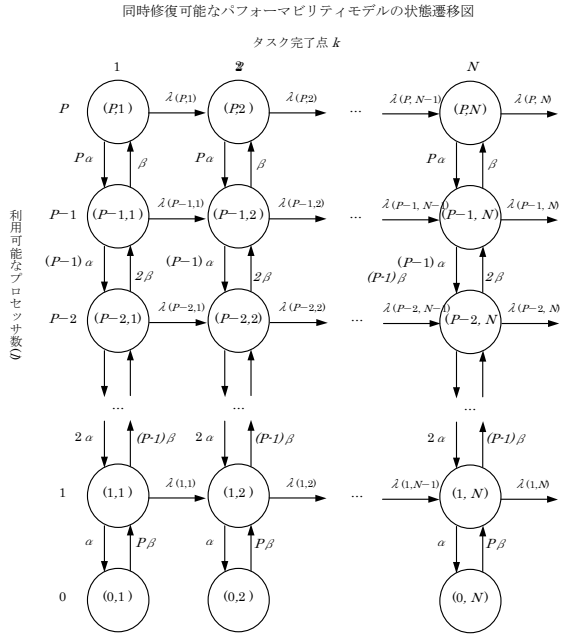

図３：同時修復可能なパフォーマビリティモデルの状態遷移図。単一修復モデルと違い、修復のレートが故障中のプロセッサ台数に比例する。

図２に、可用性とパフォーマンスのモデルを統合したパフォーマビリティモデルを示す。この図の上下に延びる矢印は可用性のモデルの故障と修復による状態遷移を示している。右向きの矢印は、並列実行中のタスクのうち一つが完了を示している（タスクが完了した時点をタスク完了点と呼ぶ）。パフォーマンスのモデルと同じようにタスクが完了するまでの時間は、利用可能なプロセッサの台数によって変動する。タスク完了による状態遷移をタスクの数だけ繰り返すとモデルの右端に到着し、すべてのタスクが完了したことを意味する。それぞれのタスク完了点間では、利用可能なプロセッサの台数がランダムに変動する。パフォーマビリティモデルのタスク完了点ごとの滞在時間の期待値を求めるには、以下の手順を用いる。

このモデルでは初期状態として、すべてのプロセッサが利用可能であると仮定する。モデルは最初の一列目を、すべてのプロセッサが利用可能として、最初のタスク完了イベントまでの時間 t を求める。計算には機械修理工のモデルから以下の無限小生成素 Q_1 を生成し、利用可能プロセッサの初期状態から確率推移式の行列 $P_1(t)$ を求める。

$$Q_1 = \begin{bmatrix} -P\alpha & P\alpha & 0 & \cdots \\ \beta & -\{(P-1)\alpha+\beta\} & (P-1)\alpha & \cdots \\ 0 & \beta & -\{(P-2)\alpha+\beta\} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

$P_1(t)$ から t 時間後の推移確率行列が求まるため、初期状態以後の利用可能プロセッサ台数の分布を求めることができる。すべてのタスク完了点について同様の手順を繰り返すことで、ジョブ実行時間などの最終的なシステムの状態を求めることが出来る。このパフォーマビリティモデルは故障したプロセッサの数にかかわらず修復される比率が一定のため、単一修復モデルと呼ぶことにする。

2.2. パフォーマビリティモデルの修正

前述したパフォーマビリティモデルは、故障したプロセッサが修復される比率が常に一定であることを仮定している。これは、複数のプロセッサが同時に故障していた場合、最初の一つの修復が完了するまで他のプロセッサの修復が待たされることを意味する。参加、不参加が比較的自由なグリッド環境においては、このモデルが旨く適用できない場合がある。そのため、先ほどのパフォーマビリティモデルから、複数同時に修復可能なモデルを派生させる（図３）。計算には出生死亡過程のモデルから以下の無限小生成素 Q_2 を生成する。

$$Q_2 = \begin{bmatrix} -P\alpha & P\alpha & 0 & \cdots \\ \beta & -\{(P-1)\alpha+\beta\} & (P-1)\alpha & \cdots \\ 0 & 2\beta & -\{(P-2)\alpha+2\beta\} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

それ以降は、単一修復モデルと同じように解決する。

3. 性能評価

二つのパフォーマビリティモデルの、グリッドに対する有効性を検証するために、グリッド環境のシミュレータによる実験結果と理論値の比較を行った。この章では、実験に使用したシミュレーションの解説をし、その後で、モデルを使用した評価結果とシュミレーション結果との比較を行う。比較は、単一修復のモデル（修復率が一定の場合）と、同時修復可能なモデルの二つにわけて行う。評価は、ジョブ全体の実行時間とタスク完了点ごとの滞在時間、タスク全体の試行回数に対するタスク失敗数の割合に注目することにした。これは、タスクやジョブの実行時間が直感的に分かりやすいことと、可用性を含めたモデルの大まかな傾向をつかむのに適しているためである。

3.1. シミュレーション

パフォーマビリティモデルから得られる値の妥当性を検証するために、仮想のグリッド環境による実験を行った。仮想のグリッド環境には専用のシミュレータを使用し、グリッド環境のタスクの実効状態とプロセッサの故障と修復をシミュレートし、タスクの完了ごとにログを出力する。パラメータには処理すべきタスクの数 N、プロセッサ数の最大数 P、タスクの実行の比率 λ、故障の比率 α、修復の比率 β である。ネットワークの延滞とサーバの逐次処理による延滞は考慮していない。このシミュレータでは、修復率が故障したプロセッサ台数に比例するモデルと、一定とするモデルを選択できるようにした。ばらつきを抑えるため、一回の実験で 500 回の繰り返し実行をして、タスク完了点ごとの滞在時間の平均値と、ジョブ全体の実行時間の平均値、総タスク実行回数に対するタスク処理中の故障率の平均値を収集した。

3.2. 単一修復モデルによる評価

はじめに、単一修復モデルによる予測値と、シミュレーションの実行結果の比較を行った。

図４と図５に、単一修復モデルのパフォーマビリティモデルを使用した理論値と実験結果を示す。

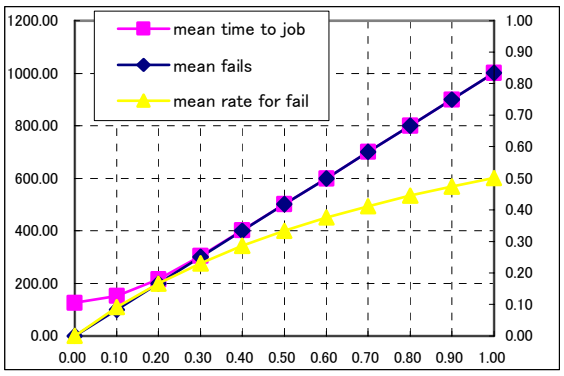

図４：単一修復モデル（パフォーマビリティモデル）
N=1000, P=8, α=0.00～1.00, β=1.0, λ=1.0
x：故障の比率（α）
y1：ジョブ実行時間　y2：タスク失敗率

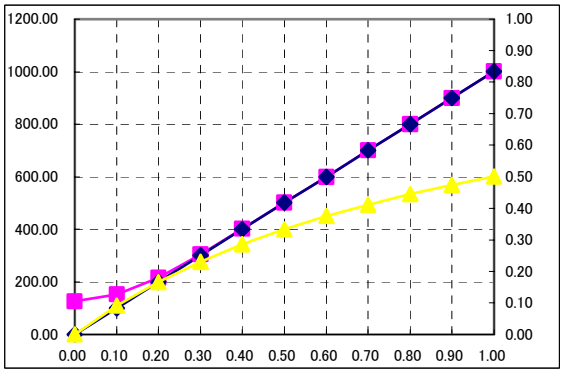

図５：単一修復モデル（シミュレーション結果）
N=1000, P=8, α=0.00～1.00, β=1.0, λ=1.0
x：故障の比率（α）
y1：ジョブ実行時間　y2：タスク失敗率

値はほぼ同一で、パフォーマビリティモデルが故障の比率に対するタスク実行時間をうまく表現していることが分かる。ここで、タスクの失敗率は、並列実行中のプロセスがタスクを処理しようとした回数に対する、故障によりタスクの処理が失敗した数の割合を示す。故障の比率が高くなるにつれて、タスクの失敗率が対数関数的に上がってゆく。

次に、タスク完了点間での滞在時間の内訳を、図６と図７に示す。

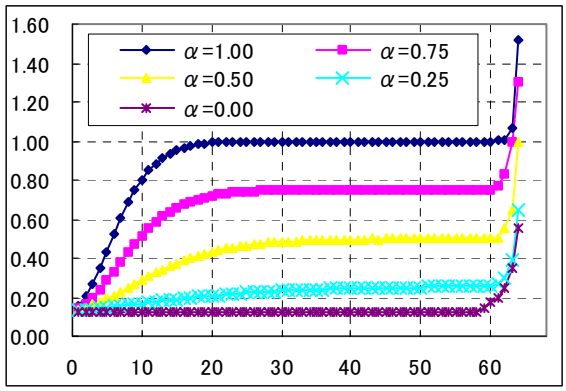

図６：単一修復モデル（パフォーマビリティモデル）
N=64, P=8, α=0.00～1.00, β=1.0, λ=1.0
x：タスク完了点(epoch)
y：タスク完了点における滞在時間

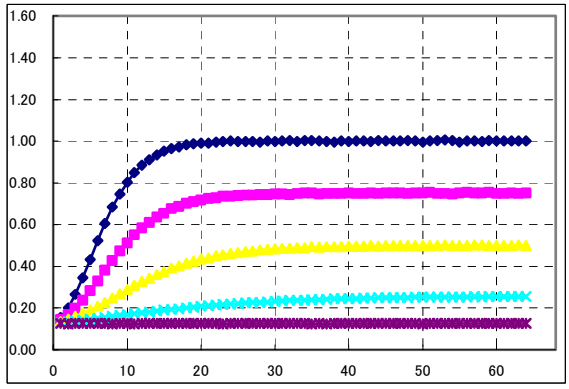

図７：単一修復モデル（シミュレーション）
N=64, P=8, α=0.00～1.00, β=1.0, λ=1.0
x：タスク完了点(epoch)
y：タスク完了点における滞在時間

興味深いことに、パフォーマビリティモデルとシミュレーションの結果で違いがみられる。図６のパフォーマビリティモデルでは滞在時間が、定常状態（20～60付近）に入ると完了間際まで安定するが、そこから完了域（60 以降）に入ると滞在時間が突発的に延びているように見える。これは、パフォーマビリティモデルの計算手順で生じた差違で、タスク完了域を分けて計算しているために生じる。タスクの数がプロセッサに対して非常に大きい場合は無視できるほど小さくなる。一方、図７のシミュレーション結果では、定常状態に入った後はタスクの実行時間が安定しているように見える。どちらの場合も、図４と図５で見られるように、全体としてのジョブ実行時間に大きな影響は出ていない。

参考のために、図８に単一修復モデルにおけるタスク完了点の到達時間を示す。これは、タスク完了点間の滞在時間を累積したものをグラフにしたものである。

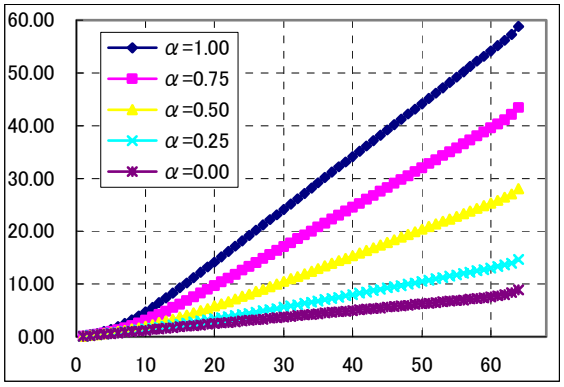

図 8：単一修復モデル（パフォーマビリティモデル）
N=64, P=8, α=0.00～1.00, β=1.0, λ=1.0
x：タスク完了点(epoch)
y：タスク完了点の到達時間

こちらでも同じように、パフォーマビリティモデルによる予測結果では、タスク完了点付近で定常状態を抜け出し、時間がかかっているように見えるが、タスク数がプロセッサ台数に比べて非常に大きい環境では無視することができる。

次に、修復の比率の変動によるジョブ実行時間の変動について比較する（図９，図１０）。

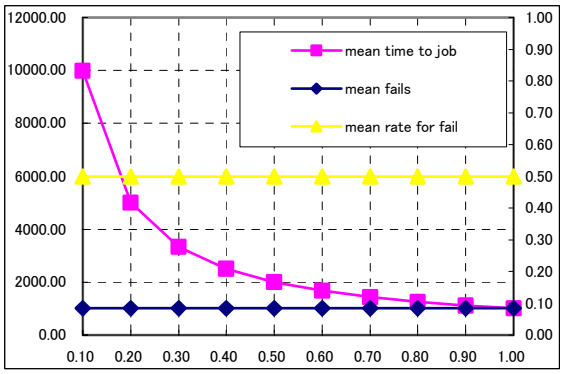

図９：単一修復モデル（パフォーマビリティモデル）
N=1000, P=8, α=1.00, β=0.00～1.0, λ=1.0
x：修復率 y1：ジョブ実行時間, y2：タスク失敗率

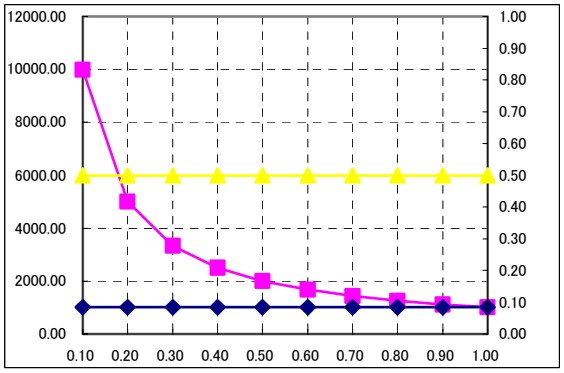

図１０：単一修復モデル（シミュレーション）
N=1000, P=8, α=1.00, β=0.00～1.0, λ=1.0
x：修復率 y1：ジョブ実行時間, y2：タスク失敗率

パフォーマビリティモデルもシミュレーション結果も、ほぼ同じ値を示している。

3.3. 同時修復可能なモデルによる評価

ここでは、故障したプロセッサが同時に修復可能な場合のパフォーマビリティモデルとシミュレーションの比較を行う。

図１１と図１２に、故障の比率を変動させた場合のジョブ実行時間を示す（図９，図１０）。

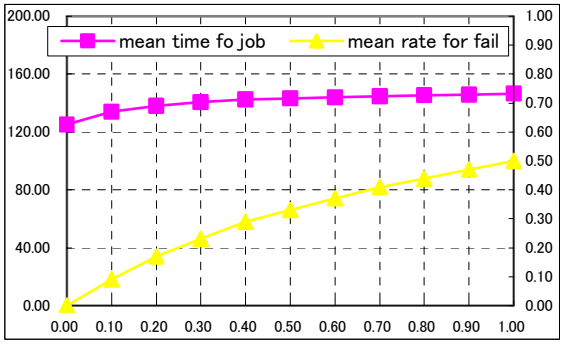

図１１：同時修復モデル（パフォーマビリティモデル）
N=1000, P=8, α=1.00, β=0.00～1.0, λ=1.0
x：故障率 y1：ジョブ実行時間, y2：タスク失敗率

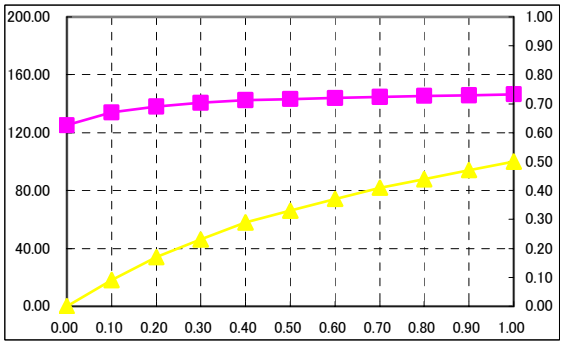

図１２：同時修復モデル（シミュレーション）
N=1000, P=8, α=1.00, β=0.00～1.0, λ=1.0
x：故障率 y1：ジョブ実行時間, y2：タスク失敗率

パフォーマビリティモデルもシミュレーション結果も、ほぼ同じ値を示している。ここで、ジョブの実行時間が、単一修復モデルと異なり上昇傾向が低くなっているのがわかる。これは、同時修復によるシステム全体の稼働率が高くなっているためだと考えられる。

次に、故障率を変動させた場合のタスク完了点に到達する時間の詳細を示す（図１３）。

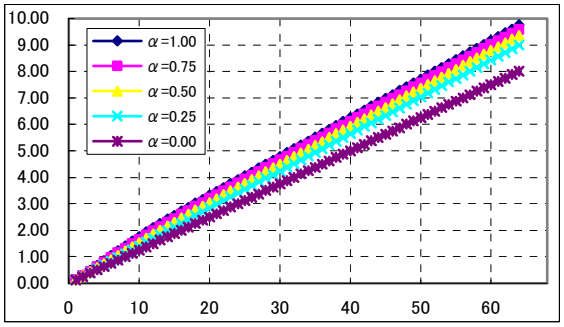

図１３：同時修復モデル（パフォーマビリティモデル）
N=64, P=8, α=0.00～1.00, β=1.0, λ=1.0
x：タスク完了点(epoch)
y：タスク完了点における滞在時間

単一修復モデルと同様に、完了域において若干の差違が見られたものの、ほぼ同じ値を示している。図８と図１３を比べた場合、単一修復モデルでは同時修復モデルと比べて修復率の比率が実行時間に大きく影響するようにみえる。これは、単一修復の場合、高い修復率を確保しなければ修理が故障追いつかなくなることを表している。

最後に修復率を変動させた場合のジョブ実行時間を示す（図１４，図１５）。

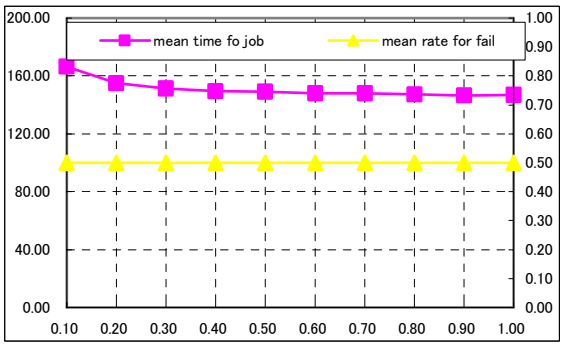

図１４：同時修復モデル（パフォーマビリティモデル）
N=1000, P=8, α=1.00, β=0.00～1.0, λ=1.0
x：修復率 y1：ジョブ実行時間, y2：タスク失敗率

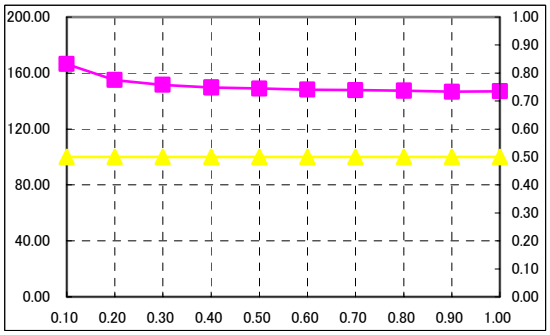

図１５：同時修復モデル（シミュレーション）
N=1000, P=8, α=1.00, β=0.00～1.0, λ=1.0
x：修復率 y1：ジョブ実行時間, y2：タスク失敗率

パフォーマビリティモデルもシミュレーション結果も、ほぼ同じ値を示している。

以上の比較から、二つのパフォーマビリティモデルは、グリッドの性能評価に対しても有効であると認められる。

4. 結論

本研究では、グリッド環境における性能評価を行うための手段として、マルコフ過程を応用した並列分散処理のための性能評価モデルを使用した。これは、タスク完了域におけるタスク実行時間の微増といった問題もあるが、プロセッサの入れ替わりの激しいグリッド環境においても十分有効であることが判明した。タスクの数がプロセッサ台数よりも十分に大きい場合において、修復の比率が一定、もしくはプロセッサ台数に比例するような場合でもモデルが有効に働き、タスクやジョブの実行時間といったシステムの評価に使用できることが判明した。元のパフォーマビリティモデルと派生モデルは故障と修復の比率をパラメータとして個々のタスクの実行時間を計算し、ジョブ全体にかかる時間の期待値や、稼働率、ワークを導き出すことができる[1]。同様に、この手法の応用により、グリッドに求められる QoS レベルを満たすために必要な修復率やプロセッサ台数の適正数など、キャパシティ計画に必要な様々な情報を得ることができる。計測できないものは制御できない。システムの複雑化、大規模化が進むにつれて無視できなくなる可用性について、より深い情報収集と研究が必要である。

参考文献

[1] Pierre M. Fiorini, Yiping Ding, "On the Performability of Computing Systems", CMG2002 *Computer Measurement Group*, Reno, NV, December 2002.

[2] G. Weerasinghe, I. Antonios, and L. Lipsky, "An Analytic Performance Model of Parallel Systems that Perform *N* tasks using *P* Processors that can Fail", *IEEE NCA 01 International Symposium on Network Computing and Applications*, Cambridge, MA, February 2002.

[3] A.O. Allen, Probability, Statistics and Queueing Theory with Computer Science Applications, volume of Computer Science and Applied Mathematics, Academic Press, Inc., ISBN 0-12-051050-2, 1976.December 2002.