ノード保守タイミングの ジョブスケジューリングへの影響評価

宇野 篤也[1,a)]　関澤 龍一[2]

概要：近年，スーパーコンピュータや PC クラスタといった HPC システムの高並列化に伴う構成部品数の増加によりシステムの故障率が高くなる傾向にある．通常の運用では，数台の計算ノードの故障がシステムの運用停止を引き起こすようなことはほとんど発生しない．しかし，ジョブスケジューリングの観点からみると故障ノード数の増加による運用への影響は無視できない．故障ノードの保守を故障ノードが発生する毎に実施することで，その影響を最小限にすることができるが，頻繁な保守作業は運用コストの面から難しい．本稿では，故障ノードの発生が運用に及ぼす影響を最小限にしつつ，保守作業の回数を減らす保守タイミングを決定する手法を検討する．今回の評価では，故障ノードの発生状況をもとに保守タイミングを決めることで，システム利用率を大きく低下させることなく，定期的に保守を行う場合よりも保守回数を減らすことが可能なことがわかった．

A Study of Job Scheduling Performance focusing on Compute Node Maintenance

Atsuya UNO[1,a)]　Ryuichi SEKIZAWA[2]

Abstract: Recently, high-performance computing systems have become larger and larger, and the failure rate of the system has also become higher. In ordinary system operation, a few node failures rarely cause the important problems. However, the increase of node failures could affect the system utilization from the point of view of the job scheduling. The daily maintenance could reduce the effect of node failures, but the frequent maintenance may also increase in the operation cost. In this paper, we propose the method to choose the maintenance schedule focusing on the compute node failures. Based on this method, we evaluated the job scheduling performance using the job scheduling simulator. As the result, our proposed method could reduce the number of the compute node maintenance without the significant performance degradation.

1. はじめに

近年，スーパーコンピュータや PC クラスタといった HPC システムは，用途の拡大や計算速度性能の向上により高並列化の傾向にある．TOP500[1] の上位にランキングしている HPC システムのコア数を見てみると，2010 年 11 月の TOP500 では，1 位から 10 位までにランキングされているシステムの平均コア数は約 15 万個であったが，5 年後の 2015 年 11 月のランキングでは，約 80 万個と 5 倍以上に増加しており，高並列化が進んでいることがわかる．システムの高並列化は構成部品数の増加を意味し，システムの故障率が高くなる原因となる．例えば，理化学研究所計算科学研究機構 (AICS) が運用を行なっている「京」は，計算ノード数が 82,944 台，コア数にすると 663,552 個と非常に高並列なシステムであり [2]，システムを構成する部品数は 100 万点以上である．

[1] 国立研究開発法人理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
[2] 富士通株式会社
FUJITSU LIMITED
[a)] uno@riken.jp

© 2016 Information Processing Society of Japan

通常の運用では，数台の計算ノードの故障がシステムの運用停止を引き起こすようなことはほとんど発生しない．しかし，ジョブスケジューリングの観点からみると故障ノード数の増加による運用への影響は無視できないものとなる．故障ノードの保守を故障ノードが発生する毎に実施することで，その影響を最小限にすることができるが，頻繁な保守作業は運用コストの面から難しい．そのため，故障ノードが運用に及ぼす影響を把握し，運用への影響が大きくなる直前に保守を実施できることが理想的である．

本稿では，故障ノードの発生が運用に及ぼす影響を最小限にしつつ，保守作業の回数を減らす保守タイミングを決定する手法を検討する．

2. ノード故障とノード保守

2.1 ノード故障とジョブスケジューリング

HPC システムの多くはバッチ形式でジョブを実行する．ユーザが投入したジョブを効率よく実行するのがジョブスケジューラの役割である．ジョブスケジューラはジョブが投入されると，あらかじめ決められたルールに従って計算ノードを割り当てる．この時，計算ノード間のネットワーク構成等を考慮して割り当てる計算ノードを決定する．特に直接網のネットワークをもつシステムでは，通信性能を確保するために，隣接した計算ノードを割り当てるといった制約が発生する場合がある．そのため，計算ノードの故障はジョブスケジューリングに大きく影響を及ぼす場合がある．故障ノードの発生がスケジューリング空間を分断し，連続して計算ノードを確保できる空間が狭まるためである．

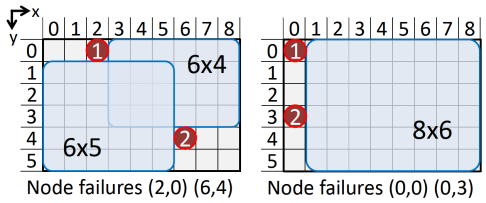

図 1 ノード故障とスケジューリング空間の関係

図 1 にノード故障とスケジューリング空間の関係を示す．ここでは 9×6 の 2 次元で各計算ノードがメッシュ接続された構成を例に説明する．赤の点 ((2,0) (6,4)) の計算ノードが故障したとする（図 1 左）．この場合，スケジューラはこれら故障した計算ノードを避けて連続した計算ノードを確保する（青の矩形）．そのため，故障ノードの発生個所によっては，確保できる矩形に大きく制限が生じることがある．図 1 左の場合，いくつか空間を確保することができるが最大空間は 6×5 となる．しかし，同じ 2 台の故障ノードでも空間全体の一番端の計算ノードが故障した場合は 8×6 となる（図 1 右）．このように故障が発生した場所で確保できる矩形に大きく差が生じることになり，単純な故障ノード数での評価は適切ではないことがわかる．

2.2 ノード保守

実際のシステム運用では，計算ノードが故障するとスケジューラが自動的に運用からその計算ノードを切り離し，保守作業を行ったあと再び運用に組み込むという作業を行う．例えば，「京」では平日は毎日保守を実施している．さらに「京」の故障率は非常に低いため，同時に故障している計算ノードは少なく，ノード故障によるシステム運用への影響は最小限に抑えられている．

保守費用を考えた場合，基本的には故障した計算ノードは全て保守を行うので，故障修理における部品代は保守回数には関係ない．しかし，保守作業の人件費等は保守回数により決まるため，保守回数を減らすことは運用コストを下げる意味でも重要である．

2.3 関連研究

ノード故障とシステム運用に関する研究はこれまでにも行われている．A. J. Oliner らは，3 次元トーラスネットワークをもつ BlueGene/L 上で，ノード故障がジョブスケジューリングに及ぼす影響を評価している [3]．ノード故障の影響を考慮したジョブスケジューリングアルゴリズムを提案し，その評価を行っている．また，後継機の BlueGene/P においても引き続き研究を行っており，ノード故障発生時のシステム利用率向上やジョブ実行待ち時間の短縮化を考慮したスケジューリングアルゴリズムについて評価を行っている [4]．

耐故障性の研究も盛んに行なわれており，システムレベルのチェックポイント/リスタート手法 [5] などが提案されている．しかし，大規模システムではチェックポイントに要する時間が長くなり実用的ではないため，投機的チェックポイント [6] など I/O の負荷を低減させる手法などが提案されている．

本稿は，保守の観点からシステム利用への影響を最小限にする手法に関する研究であり，システムの効率的な運用という点は共通しているが着眼点が異なる．また，これら他の手法と同時にシステム運用へ適用可能な手法である．

3. 評価環境

今回の評価で使用したジョブスケジューラシミュレータについて述べる．シミュレータは，あらかじめ作成したジョブミックスとノード故障パターンを読み込み，各ジョブに設定された投入時刻に従って，ジョブの実行処理をシミュレートする．また，ノード故障パターンに従って，ノード故障を発生させる．この時，実行中のジョブがあった場合には当該ジョブの実行を中止し，故障ノードをスケジューリングの対象外とする．実行が中断されたジョブは再スケジューリングとなる．なお，除外された計算ノード

© 2016 Information Processing Society of Japan

は保守実施後にスケジューリング対象となる．

今回使用したスケジューリングアルゴリズムはFCFS[*1]とBackfill[*2]で，ネットワークトポロジーの制約を想定して連続した1次元の計算ノードをジョブに割り当てることとした．また，システム全体のノード数は768台とし，使用したジョブミックスは「京」で実行されたジョブの統計情報をもとに生成した [7].

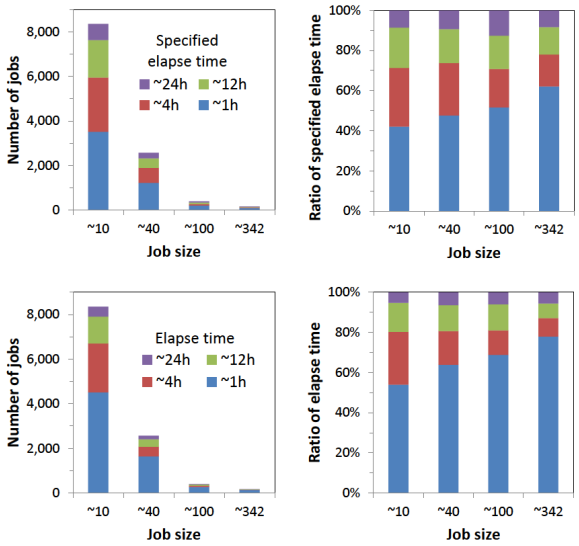

図 2 シミュレーションで使用したジョブセットの統計情報

図 2 に今回使用したジョブミックスの統計情報を示す．ジョブの最大ノード数は342ノード，ユーザがジョブを投入する際に指定する指定経過時間は最長で24時間とした．図2からわかるように，規模の小さいジョブの割合が多く，指定経過時間と実際の経過時間の差が大きいという特徴がある．ジョブミックスは，ジョブ投入期間を28日間（4週間），密度を80%として生成した．ジョブミックスの密度とは，システムの全計算資源に対する全ジョブの実際に計算ノードを使用したノード時間積の総和の割合である．つまり，80%の密度をもったジョブミックスでシミュレーションを実行した場合，システム利用率は最大でも80%となる．

図 3 にシミュレーションで使用したノード故障パターンを示す．縦軸がノード故障が発生した個所（全ノード数を1として計算した場合の位置）を，横軸が発生日時を表している．5セット分を示しているため，5種類の記号がプロットされている．ジョブ投入期間中に1日に約1件のノード故障が発生する確率で生成しており，複数回ノード故障が発生する日や，一回も発生しない日もある．

ジョブミックスは10セットを，ノード故障パターンは5セットをそれぞれ作成した．本稿での評価では，これら

[*1] First-Come and First-Served：ジョブの投入順に優先順位を決定するアルゴリズム
[*2] Backfill：空ノードがある場合にジョブの実行順序を入れ替えてシステム利用率を改善するアルゴリズム

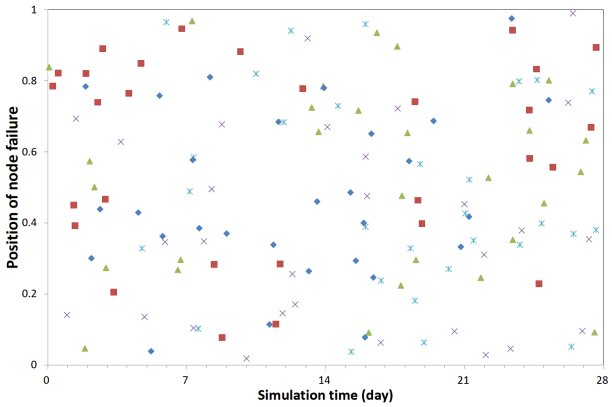

図 3 ノード故障パターン（5セット分）

ジョブミックスとノード故障パターンを組み合わせ，1条件に対し $5 \times 10 = 50$ パターンのシミュレーションを実施した．ジョブ実行に関しては集計期間内に実行が開始されたジョブの実行待ち時間とジョブ全体に対する実行割合を，システムの運用効率に関してはシステム利用率と保守の実施回数を比較している．集計期間はシミュレーション開始後2日目からジョブ投入が終了する28日目までの27日間とした．最初の1日目を除外したのは，シミュレーション開始直後は投入されたジョブが少なく評価に適さないためである．なお，システム利用率の計算では，ジョブの実行中にノード障害が発生した場合は，ジョブ実行開始からノード障害発生までの間は計算ノードは使われていなかったものとして計算している．

4. 一定間隔の保守

まずは，計算ノードの故障数に関係なく一定間隔で保守を実施する場合のジョブスケジューリングへの影響について評価を行った．

一定間隔の保守のシミュレーションでは，保守間隔として，1日毎 (D=1)，4日毎 (D=4)，7日毎 (D=7) に保守を行った場合についてシミュレーションを実施した．保守はシミュレーション内時間の0時に実施するものとした (1日毎の場合はシミュレーション開始から24時間目，48時間目，72時間目…となる)．参考までに，同じ条件でノード故障が発生しない場合 (D=0) についても評価を行った．

図 4 にシステム利用率を，**図 5** にジョブの実行待ち時間

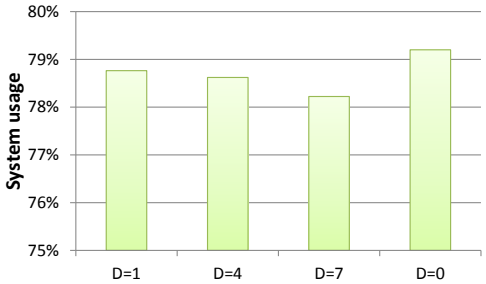

図 4 一定間隔で保守を実施した場合のシステム利用率

© 2016 Information Processing Society of Japan

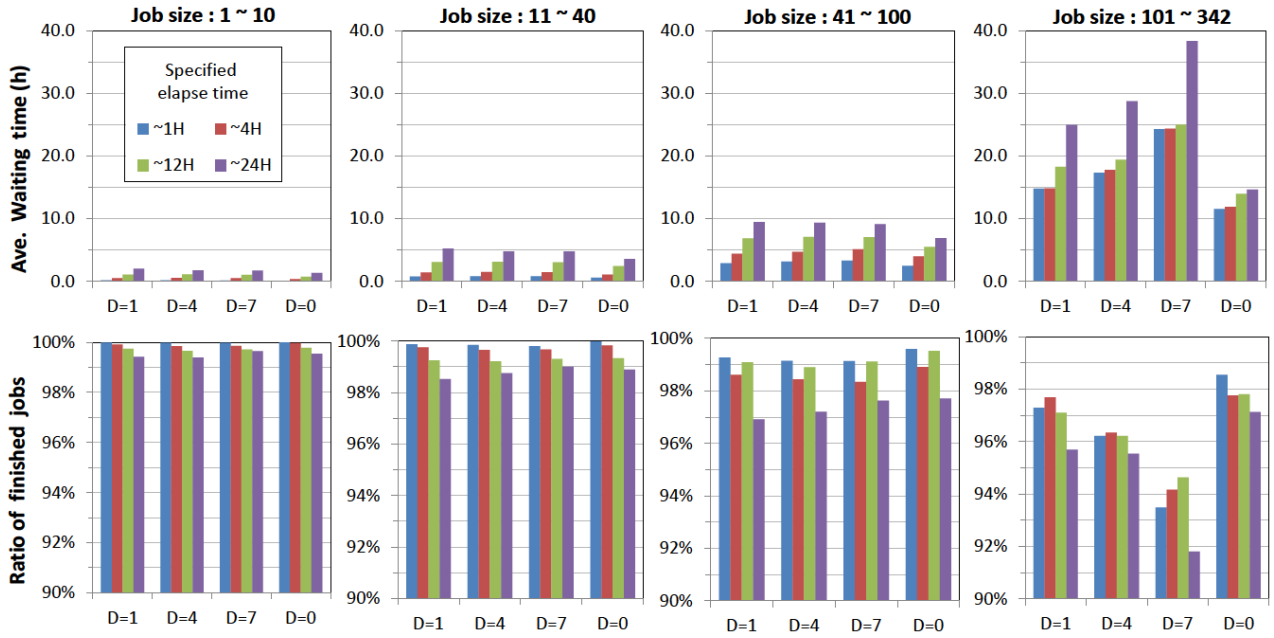

図 5 一定間隔で保守を実施した場合の実行待ち時間とジョブの終了割合

とジョブの終了割合をそれぞれ示す．保守間隔が広がるにつれシステム利用率が低下していることがわかる．ジョブの実行待ち時間では，100 ノード以下のジョブの待ち時間はほとんど変化していないが，100 ノードより大きいジョブ，特に指定経過時間が 12 時間より長いジョブの待ち時間が長くなっている．また，ジョブの終了割合についても 100 ノード以下のジョブはほとんど変わりがないが，100 ノードより大きいジョブ，特に指定経過時間が 12 時間より長いジョブの終了率が低くなっている．

以上の結果から，保守間隔が長くなることで同時に故障しているノード数が増加し，規模の大きいジョブが実行されにくい状況になることがわかる．このことから，ノード故障によるスケジューリングへの影響を最小限にしつつ保守回数を減らす方法として，ノード故障によるスケジューリング空間の変化に着目し保守の実行タイミングを決定することとした．

5. 故障状況を考慮した保守

5.1 故障状況の評価

一定間隔で保守を実施した場合の評価結果から，スケジューリング空間の大きさがジョブスケジューリングに大きな影響を及ぼすことが分かった．そこでスケジューリング空間を評価する方法として以下の方法について検討した．

- スケジューリング空間の最大値
- スケジューリング可能な組み合わせ数

なお，簡略化のため，ここでは 1 次元のスケジューリング空間を対象とする．

5.1.1 スケジューリング空間の最大値

この評価方法では，ノード故障の発生により分割されたスケジューリング空間における最大空間の大きさを評価値とする．評価式 E_s を式 (1) に示す．

$$E_i = \frac{S_i}{S_{max}} \qquad E_s = Max(E_i) \tag{1}$$

ここで，S_i はノード故障により分割された個々のスケジューリング空間の大きさを，S_{max} はノード故障が発生していない状態でのスケジューリング空間の大きさをそれぞれ表す．

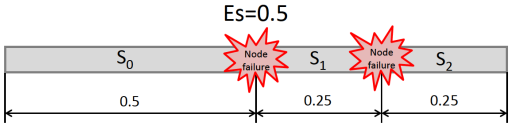

図 6 スケジューリング空間の最大値による評価の例

図 6 にスケジューリング空間で計算ノードが故障した場合の例を示す．スケジューリング空間の中央に位置する計算ノードが故障した場合，スケジューリング空間は 2 分割されるので $E_s = 0.5$ となる．さらに分割された片方のスケジューリング空間の中央に位置する計算ノードが故障した場合，そのスケジューリング空間の評価値は $E_1 = 0.25$ となる．しかし，システム全体の最大スケジューリング空間は変化しないので，$E_s = 0.5$ のままである．

5.1.2 スケジューリング可能な組み合わせ数

この評価方法では，あるジョブがあるスケジューリング空間においてスケジューリング可能な組み合わせ数をもとに評価値を求める．前述のスケジューリング空間の最大値との大きな違いは，特定のスケジューリング空間のみに着目するのではなく，分割された全てのスケジューリング空間を対象にする点である．評価式 E_c を式 (2) に示す．

© 2016 Information Processing Society of Japan

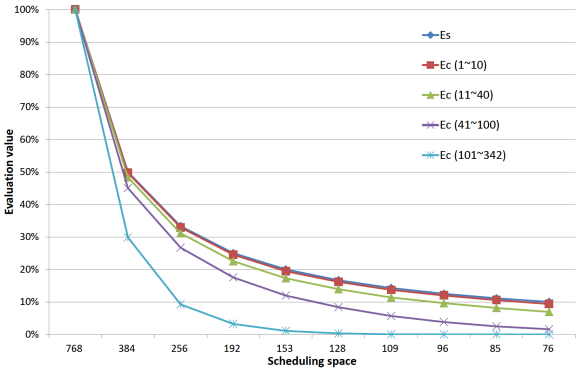

図 7 スケジューリング空間と評価値の関係

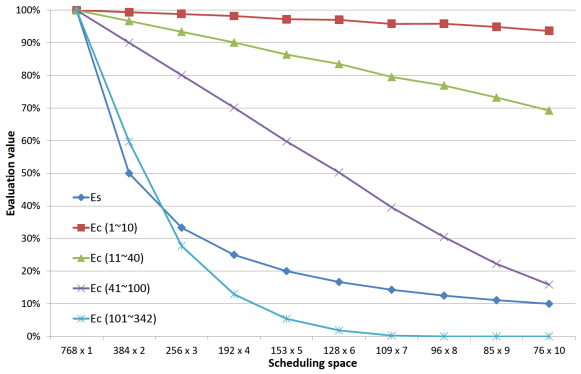

図 8 等分割した場合のスケジューリング空間と評価値の関係

$$E_n = \sum_{i=j}^{k} S_{n-i+1} C_1 \qquad E_c = \frac{\sum_{i=0}^{k} E_i}{E_{max}} \qquad (2)$$

ここで，E_n はノード故障により分割されたスケジューリング空間におけるサイズ j〜k のジョブがスケジューリング可能な組み合わせの総和を，E_{max} はノード故障が発生していない状態での E_n を，S_n はスケジューリング空間の大きさ（ここでは 1 次元なのでノード数に一致）をそれぞれ表している．なお，$S_n < i$ の時はスケジューリングできないので $E_n = 0$ とする．

式 (2) からわかるように，規模が小さくなるほどスケジューリング可能となる組み合わせ数が増え，空間分割の影響を受けにくくなる．そのため，実行可能な全てのジョブの E_n の総和で E_c を求めると，規模の小さいジョブが多い場合にはそれらが支配的になってしまうため，ジョブの規模別に複数のグループを作成しグループ単位で評価値を求めることとした．なお，スケジューリングの組み合わせ数はノード規模に関して包含関係にあるため，規模の大きいグループの評価値を利用した場合，それより小さい規模のグループも同様に影響を受けることになる．

5.2 評価式の評価

本稿で提案する評価式とスケジューリング空間の大きさの関係について評価を行った．

図 7，**図 8** にスケジューリング空間の分割状況と各評価値の関係を示す．図 7 は分割したスケジューリング空間のノード数を 768,384,…,76 と変化させた時の個々の分割したスケジューリング空間の各評価値を，図 8 は 768 ノードのスケジューリング空間を等分割した場合のシステム全体の評価値（全ての分割されたスケジューリング空間を対象にした評価値）をそれぞれ示している．ここではスケジューリング空間は一次元とした．

図 7 からわかるように，規模の小さいグループはスケジューリング空間が小さくなっても E_s との差は小さい．一方，規模の大きいグループは E_s と比較して急激に評価値が低くなる傾向にある．図 8 では，E_s はスケジューリング空間の分割に合わせて減少するが，規模の小さいグループは E_s ほど分割の影響は受けていない．規模の大きいジョブ，特に 101〜342 ノードジョブのグループは分割の影響を大きく受け，E_c と E_s が逆転する現象が起きている．このことから，E_s で保守タイミングを決定する場合，故障ノード数が増えスケジューリング空間の分割が進むと 101〜342 ノードのジョブは E_c で決定する場合より実行されにくい状況になることが予想される．

図 9 に一定間隔で保守を実施した場合の E_s と E_c の推移を示す．同一のジョブセットとノード故障パターンを使用し，$D = 1, 4, 7$ の各保守間隔についてシミュレーションを行った．E_c は 101〜342 ノードジョブのグループの評価値である．図 8 と同様に全般的に E_c は E_s よりも低い値になっている．また，保守間隔が長くなるにつれ E_c も低くなり，101〜342 ノードジョブが実行されにくい状況になっている．これは保守間隔が長くなるにつれ規模の大きいジョブが実行されにくくなった図 5 の結果と一致する．

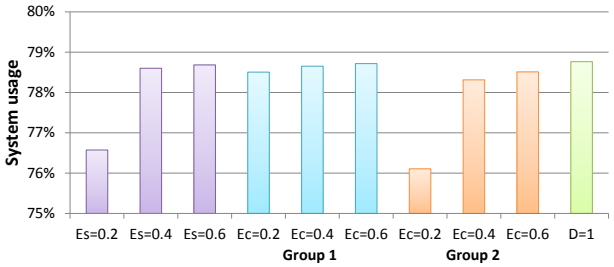

図 10 故障状況を考慮して保守を実施した場合のシステム利用率

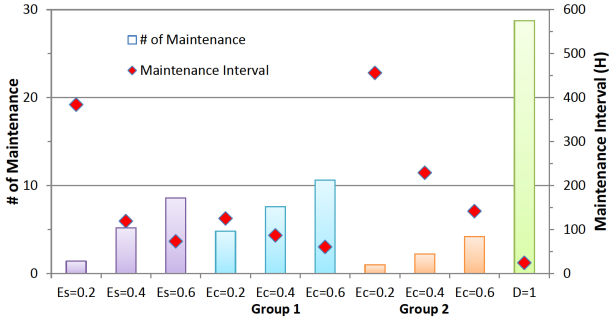

図 11 故障状況を考慮して保守を実施した場合の保守回数と保守間隔

© 2016 Information Processing Society of Japan

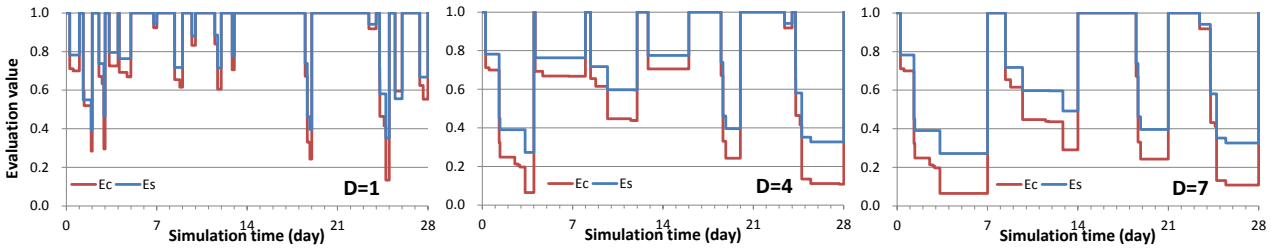

図 9 一定間隔で保守を実施した場合の E_s と E_c の推移

5.3 シミュレーションによる評価

前述の評価式に基づいて保守タイミングを決定した場合のシミュレーションを行った．E_c は，グループ 1（101〜342 ノードジョブ）とグループ 2（41〜100 ノードジョブ）の 2 グループの評価値を使用した．E_s，E_c それぞれの評価値が 20%，40%，60%になった場合に保守を実施する条件で評価した．なお，保守の実施時刻はノード故障発生時刻に関係なく，シミュレーション内時間で評価値の条件を満たす故障が発生した翌日の 0 時とした．

図 10 にシステム利用率を，**図 11** に保守の平均実行回数と平均保守間隔を，**図 12** にジョブの実行待ち時間とジョブの終了割合をそれぞれ示す．

システム利用率は，$E_s = 0.2$ と $E_c = 0.2$（グループ 2）を除いて毎日保守を実行する場合と比べて大きな差はでていない．一方，保守の平均実行回数を比較すると，E_s，E_c ともに毎日保守を実行する場合より大幅に削減できていることがわかる．$E_s = 0.4$ の場合と $E_c = 0.4$（グループ 1）の場合を比較すると，システム利用率はほぼ同じだが，保守回数が $E_c = 0.4$（グループ 1）の方が多くなっている．しかし，ジョブの終了割合を比較すると，$E_c = 0.4$（グループ 1）は 101〜342 ノードジョブが高くなっている．一方，$E_s = 0.4$ は 41〜100 ノードジョブが高くなっており，101〜342 ノードジョブの代わりに実行されたことがわかる．このことから，$E_s = 0.4$ では 101〜342 ノードジョブにとっては保守タイミングが遅いことがわかる．これは，評価式の考察結果と一致する．特に $E_s = 0.2$ では明らかに保守タイミングが遅く，101〜342 ノードジョブの実行が大きく妨げられている．

次にグループ 1 とグループ 2 について比較した．システム利用率および保守回数は $E_c = 0.2$（グループ 1）の場合と $E_c = 0.6$（グループ 2）の場合がほぼ同じ結果となっている．しかし，ジョブの実行待ち時間およびジョブの終了割合をみると，$E_c = 0.6$（グループ 2）の場合の 101〜342 ノードジョブの結果が $E_c = 0.2$（グループ 1）の場合よりも悪くなっている．図 8 をみると，E_c（グループ 2）が 0.6 をとるとき E_c（グループ 1）は約 0.05 となっている．実際には等分割にはならないので図 8 をもとに単純に比較することはできないが，$E_c = 0.6$（グループ 2）の場合，101〜342 ノードジョブは $E_c = 0.2$（グループ 1）の場合よりも実行されにくい状況になると考えられる．

以上の結果から，評価値 E_c を用いることで，システム利用率を大きく低下させることなく，定期的に保守を行う場合よりも保守回数を減らすことが可能であることがわかった．また，同じ評価値 E_c でも，評価対象のジョブサイズにより，ジョブの実行状況をある程度コントロールすることが出来ることもわかった．

今回の評価では特定の評価対象の評価値を使用したが，投入されているジョブの状況に応じて評価対象を変更することで，さらなる保守タイミングの最適化が可能になると考えている．

6. おわりに

本稿では，故障ノードの発生状況をもとに保守タイミングを決める手法について述べた．本手法では，保守タイミングをジョブがスケジューリング可能な組み合わせ数にもとづいて決定する．この評価方法では，単純な故障ノード数やスケジューリング空間の最大値による評価と異なり，ノード故障により分割されるスケジューリング空間全体を評価対象としている．シミュレーションによる評価では，本手法を用いることでシステム利用率を大きく低下させることなく，定期的に保守を行う場合よりも保守回数を大きく減らすことが可能であった．また，評価基準となるジョブサイズを変更することにより，ジョブの実行状況をある程度コントロールすることも可能であった．

今回の評価では 1 次元のノード構成で評価したが，直接網を採用するシステムの多くは多次元で構成されており，実際の運用に適用するためには，多次元におけるスケジューリングの組み合わせを考える必要がある．また，投入されているジョブの状況を考慮し評価基準を決めることで，さらなる最適化が可能になると考えており，引き続き検討を行っていく．

参考文献

[1] TOP500: TOP500 Supercomputer Sites, Top500.org (online), available from (http://www.top500.org) (accessed 2015-5-20).

[2] Yamamoto,K., Uno,A., Murai,H., Tsukamoto,T., Shoji,F., Matsui,S., Sekizawa,R., Sueyasu,F., Uchiyama,H., Okamoto,M., Ohgushi,N., Takashina,K., Wakabayashi,D. ,Taguchi,Y., Yokokawa,M. : The K

© 2016 Information Processing Society of Japan

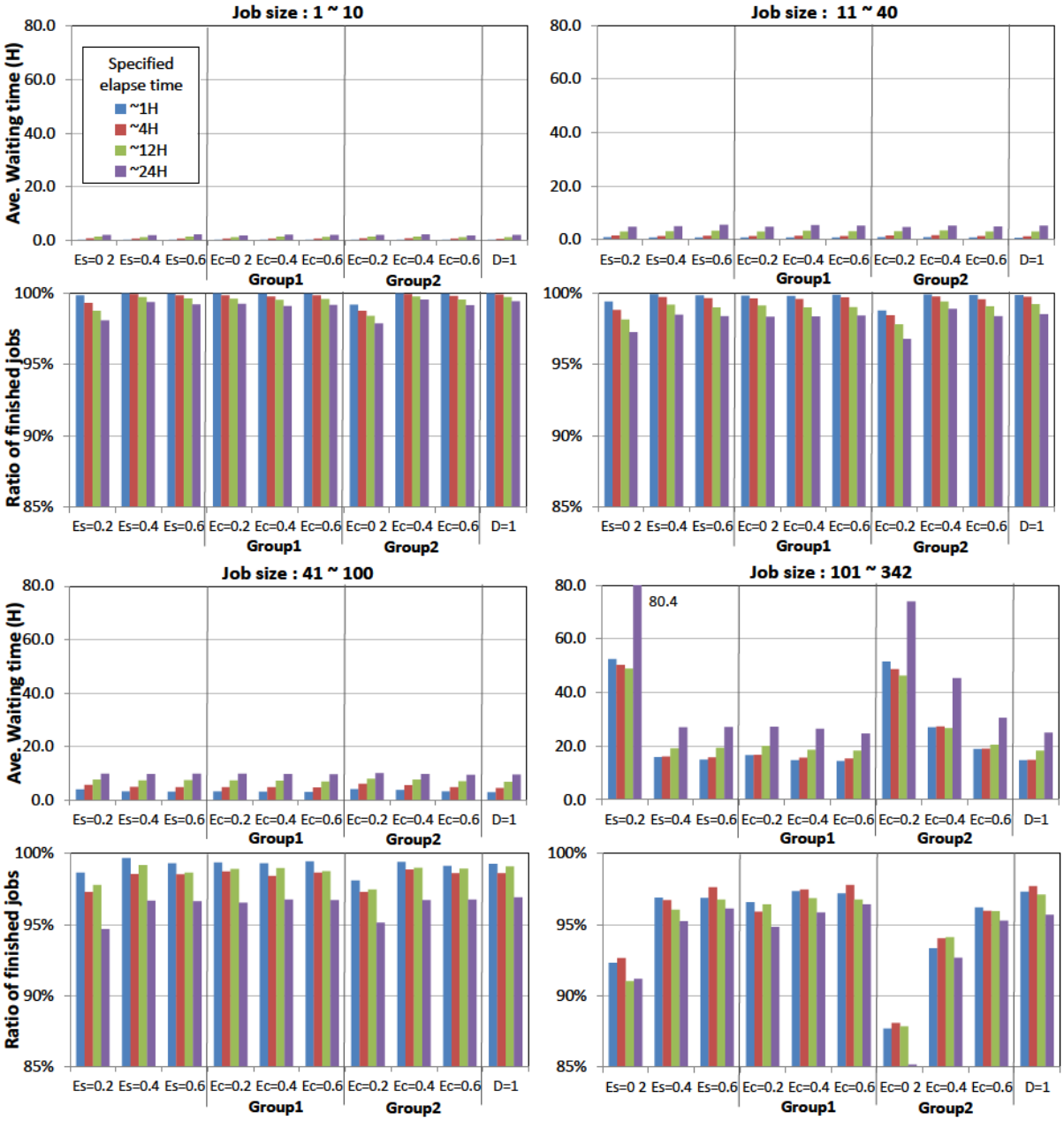

図 12 故障状況を考慮して保守を実施した場合の実行待ち時間とジョブの終了割合

computer Operations: Experiences and Statistics., International Conference on Computational Science ICCS2014, pp. 576-585 (2014).

[3] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. : Fault-aware Job Scheduling for BlueGene/L Systems, Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS04), p.64 (2004).

[4] W. Tang, Z. Lan, N. Desai, and D. Buettner : Fault-aware,utility-based job scheduling on Blue Gene/P systems, Proc. IEEE Int'l Conf. Cluster Computing, pp.1-10 (2009).

[5] J. S. Plank, Y. Chen, and K. Li: CLIP A Checkpointing Tool for Message-Passing Parallel Programs, Proc. of the 1997 ACM/IEEE Conference on SuperComputing (SC'97), pp.1–11 (1997).

[6] I. Yamagata, S. Matsuoka, H. Jitsumoto, and H. Nakada: Speculative Checkpointing: Exploiting Temporal Affinity of Memory Operation, HPC Asia 2009, pp.390–396 (2009).

[7] 宇野篤也, 関澤龍一, 山本啓二, 若林大輔, 庄司文由：「京」上のジョブの分析とジョブミックス生成手法の提案, 情報処理, Summer United Workshops on Parallel, Distributed and Cooperative Processing, SWoPP2015(2015).

© 2016 Information Processing Society of Japan