Vol.2014-HPC-146 No.3
2014/10/2

区間スケジューリングを用いたジョブスケジューリングの性能評価

山本 啓二[1,a)]　宇野 篤也[1,b)]　関澤 龍一[2,c)]　若林 大輔[3,d)]　庄司 文由[1,e)]

概要：スーパーコンピュータや PC クラスタなどで大小さまざまなジョブを効率よく実行するにはジョブスケジューラの機能が重要である．特に数万ノードの高並列システムでは同時に処理するジョブ数は千近くになる場合もあり，効率的なジョブスケジューリングは，システムの有効活用の観点からも必要不可欠である．効率よくスケジューリングを行う一つの方法として，我々は区間ジョブスケジューリング法を提案する．提案手法では，時間軸方向をいくつかのスケジュール区間に分割し，個々のジョブをそのスケジュール区間単位に切り上げてスケジューリングする．スケジュール区間の長さを変化させることで，スケジューリングのコストと効率をコントロールすることができる．本稿では，区間の長さとスケジューリング効率の関係について評価した結果を報告する．

1. はじめに

スーパーコンピュータや大規模なクラスタシステムではバッチジョブの形で大小様々なジョブが投入される．ジョブスケジューラは，時間規模やノード規模を考慮してジョブに計算資源を割り当てる．システムの利用率はジョブスケジューラの資源割り当てアルゴリズムに左右され，システムの有効活用の観点からもジョブスケジューラには高い利用効率が求められている．TOP500[1] にランキングされるスーパーコンピュータはノード数も数万から数十万と非常に大きく，投入されるジョブの数も多い．例えば 2014 年 6 月の TOP500 ランキングで 4 位のスーパーコンピュータ「京」（以下「京」）の場合，2012 年 10 月から 2013 年 9 月までに一月あたりに少ない月でも約 20,000 ジョブ，多い月で約 60,000 ジョブが実行されている [2], [3].

本稿では効率よくスケジューリングを行う方法として区間ジョブスケジューリング法を提案する．提案手法は時間軸方向をいくつかの区間に区切り，その区間単位でジョブのスケジューリングを行う方法である．時間軸を長く区切ればスケジューリングに要する計算時間も短くなり，大量のジョブを処理できるようになる．逆に時間軸を短く区切るとスケジューリングに要する計算時間は長くなり，大規模システムではジョブスケジューリングが追いつかないことがある．本手法では，時間軸方向の区間の区切り方をどのようにするかによって，スケジューリングに要する計算時間およびシステム利用率，ジョブの実行状況が変化する．2 章では提案手法について述べ，3 章では提案手法を使ってジョブをスケジューリングした場合のシステム利用率やジョブの待ち時間等の変化について評価する．4 章でジョブスケジューラに関する関連研究を紹介し，5 章でまとめと今後の課題について述べる．

2. 区間ジョブスケジューリング

通常，ジョブスケジューラはジョブが投入された際に，そのジョブが要求するノード数や経過時間およびジョブの実行優先度を考慮してスケジューリングを行う．提案手法では時間軸方向をいくつかのスケジュール区間に分割し，ジョブの指定経過時間をスケジュール区間単位に切り上げてスケジューリングを行う．スケジュール区間の導入によって，大きく変化するのはスケジューリングに要する時間である．例えばスケジュール区間が 1 秒単位の場合に指定経過時間が 5 分 (300 秒) のジョブをスケジューリングすることを考える．この場合，スケジューラは時間軸方向にスケジュール区間で連続した 300 個の空き領域を探索する必要がある．一方でスケジュール区間が 1 分単位の場合には，スケジュール区間で連続した 5 個 (5 分) の空き領域を

[1] 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
[3] 株式会社富士通ソーシアルサイエンスラボラトリ
Fujitsu Social Science Laboratory Limited
[a)] keiji.yamamoto@riken.jp
[b)] uno@riken.jp
[c)] r_sekizawa@jp.fujitsu.com
[d)] waka-d@jp.fujitsu.com
[e)] shoji@riken.jp

探索するだけでよい．探索アルゴリズムにも依存するが，スケジューリング区間が 1 秒の場合と 1 分の場合とで 60 倍もの探索空間の広さに差が生じることになる．

「京」で 1 分単位のスケジュール区間で 24 時間分のスケジューリングを行う場合を考えてみる．「京」は 82,944 ノードから構成されるため，探索空間はノード数 (82,944) と時間 (60 × 24) を掛けた約 1 億 2 千万個となる．この膨大な空間をスケジューリングの度に探索することになる．ジョブのスケジューリングは基本的にはジョブの投入時と終了時に行われ，ジョブ数が多ければスケジューリングの契機も増えることになる．探索空間の大きさはスケジューリングに要する時間を左右するため，大規模な環境では適切な探索空間を決めることは非常に重要である．

図 1 に提案手法のジョブスケジューリングの状況を示す．縦軸はノード横軸は時間をそれぞれ表し，原点は現在時刻である．矩形はジョブを示し，矩形上の数値はジョブの投入順である．ジョブは投入順に処理される (First-Come and First-Served) ものとし，ジョブには連続したノードが割り当てられるものとする．

図 1 の (1) は理想的なスケジューリングを示す．ジョブ 1 の終了予定時刻の次の瞬間にジョブ 5 は動き始め，ジョブ 3 の終了予定時刻の次の瞬間にジョブ 6 は動き始める．(2) は提案手法の粗密組み合わせ区間を使ったスケジューリングで，スケジューリング期間の前半分を短く区切り，後半分を長く区切ったものである．(1) と同様にジョブ 1 の指定経過時間以降にジョブ 5 が実行開始するが，その開始はスケジューリング区間の開始タイミングと同じとなる．そのためジョブ 1 と 5 の間にはスケジューリング区間以下のノード空き時間ができることになる．(3) はスケジューリング期間とスケジューリング区間を等しくしたもので，もっとも粗いスケジューリング区間である．ジョブ 1-4 は，指定経過時間の長さに関係なくスケジューリング期間のすべてを使ってノードを確保する．ジョブ 5,6 は，スケジューリング期間内にジョブ 1-4 がスケジューリングされていてノードに空きがないため，スケジューリングの対象外となる．

このようにスケジューリング区間の設定の仕方によって探索空間の大きさ，スケジューリングに要する時間，システムの利用率，ジョブの待ち時間等に差ができる．次章では，区間の長さとシステムの利用率の関係などいくつかの指標を用いて提案手法を評価する．

3. 評価

提案手法をシステムの利用率とジョブの待ち時間およびスケジューリングコストの 3 点から評価した．まず評価に用いたジョブミックスの生成手法および提案手法を実装したジョブスケジューラシミュレータについて説明し，続いてシミュレーション結果から提案手法の優劣について述べる．

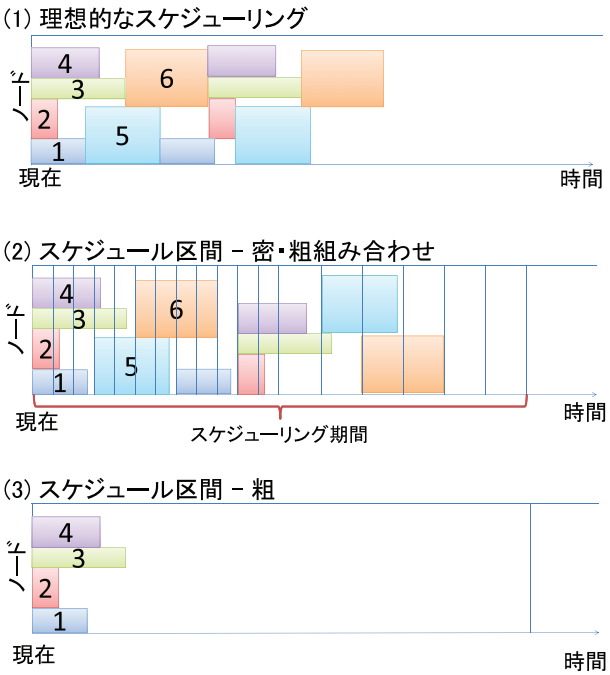

図 1 スケジュール区間の粒度によるジョブスケジューリングの状況

3.1 評価環境

3.1.1 ジョブミックス

ジョブミックスは様々なジョブの集合体で，個々のジョブについて，ジョブの投入時刻 (投入間隔)，ジョブのノード数，ジョブの指定経過時間，ジョブの実行時間の 4 つのパラメータをもつ．ジョブスケジューラの評価をする場合はひとつのジョブミックスだけではなく，様々なジョブミックスを使って統計的に評価をする必要がある．本稿では，このジョブミックスを統計的に生成した．

統計元として「京」で 2013 年 12 月から 2014 年 5 月までに投入されたジョブの情報を用いた．「京」は通常の運用では 1-384 ノードまでのジョブを投入できる small リソースグループと 385-36,864 ノードまでのジョブを投入できる large リソースグループに分かれている．「京」は 82,944 ノードから構成されるが，そのうち 9,216 ノードが small リソースグループに割り当てられ，残りの 73,728 ノードが large リソースグループに割り当てられている．評価では，large リソースグループに投入されたジョブにのみ注目し，これらのジョブをジョブの指定経過時間が 1 時間まで，4 時間まで，12 時間まで，24 時間までに分類し，それぞれについてジョブのノード数が 385-1,080 ノードまで，4,320 ノードまで，10,800 ノードまで，36,864 ノードまでに分類した．それぞれについて，指定経過時間やノード数および実際の実行時間やジョブ数，ジョブの投入頻度の平均値や分散を求めた．これらの統計情報を使って，ジョブスケジューラシミュレータで使用するジョブミックスを作成した．

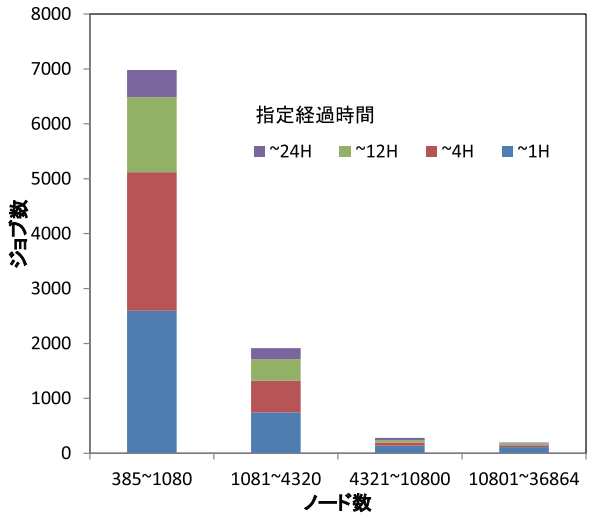

図 2 21 日分のジョブミックスの内訳

生成したジョブミックスの一例を**図 2** に示す．図 2 は 21 日間に投入されるジョブをノード数と指定経過時間で分類したものである．1,080 ノード以下のジョブが全体の約 75%を占め，1,081 から 4,320 ノード以下のジョブが約 20%を占めている．また指定経過時間が 1 時間以内のジョブが 38%を占め，4 時間以内のジョブ，12 時間以内のジョブがそれぞれ 34%，20%を占めている．本評価で用いるジョブミックスはすべて統計的には図 2 に示す特性を持つ．

3.1.2 ジョブスケジューラシミュレータ

提案手法を用いたジョブスケジューラシミュレータを実装した．シミュレータはジョブミックスを読み込み，ジョブの投入時刻に従ってジョブの投入，ノードの確保，ジョブ実行/終了，ノードの開放などの一連のジョブスケジューラの動作をシミュレーションする．このシミュレータは 5 分毎に使用ノード数，実行中のジョブ数，キュー状態のジョブ数を出力する．また，ジョブ毎にジョブがキュー状態から実行へ遷移した時刻およびジョブが確保したノードを出力する．システムの利用率は 5 分毎に得られる使用ノード数の値から計算する．ジョブの実行待ち時間はジョブの投入時刻とジョブが実行へ遷移した時刻から計算する．ジョブスケジューラはノードが 1 次元に連続しているものと仮定し，ジョブには連続したノードのみ割り当て可能とした．スケジューリングアルゴリズムはジョブの投入順にスケジューリングを行う FCFS と，空きノードがある場合にジョブの実行順序を変更してスケジューリングするバックフィルを用いた．

3.1.3 評価パラメータ

スケジュール区間として**図 3** に示す A,B,C の 3 パターンについて評価を行った．パターン A は 1 時間までは 10 分のスケジュール区間で，12 時間までは 30 分の区間で，24 時間までは 2 時間の区間で，36 時間までは 4 時間の区間で，最後の 60 時間までは 8 時間の区間でスケジューリングを行う．近い未来は短い区間を用い，遠い未来は長い区間を用いる粗密を組み合わせたパターンである．パターン B は全期間で 1 時間のスケジュール区間を用いたスケジューリングを行う．パターン C は全期間で唯一のスケジュール区間を用いたスケジューリングを行う．つまりスケジューリング期間と等しい 60 時間の区間を用いている．スケジューリング区間の数は，A,B,C それぞれ 40 個，60 個，1 個である．

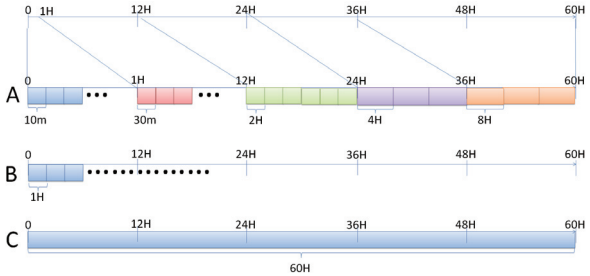

図 3 評価パターンのスケジュール区間

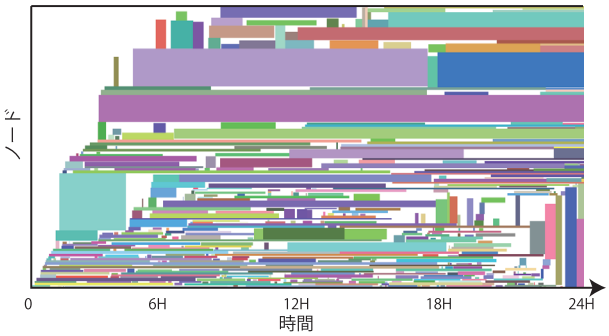

図 4 スケジュール状態の例

シミュレーションは 21 日間実行し，スケジューリング期間は 60 時間とした．シミュレーション時間の短縮のため，システムのノード数を「京」のフルノード数である 82,944 ノードを 768 ノードに縮小し，ジョブミックスのノード数も同様の比率で縮小した．1 回のスケジューリングに要する時間は基本的にノード数とスケジューリング区間の数に比例する．つまり，パターン C のスケジューリングコストが一番小さく，パターン B のコストが一番大きくなる．ジョブミックスは 50 セット用意し，各パターンそれぞれ 50 回のシミュレーションを行い，平均値を評価結果とした．

実際にパターン A でシミュレーションを行った時のスケジュール状態を**図 4** に示す．横軸はシミュレーション開始から 24 時間経過までの時間を，縦軸はノードを示している．ジョブは図の下に位置するノードから割り当てられている．大小様々な矩形はジョブを示しており，空白の領域はノードにジョブが割り当てられていないことを表している．

3.2 システム利用率

図 5 に 2 日目から 21 日目までのシステムの利用率を示

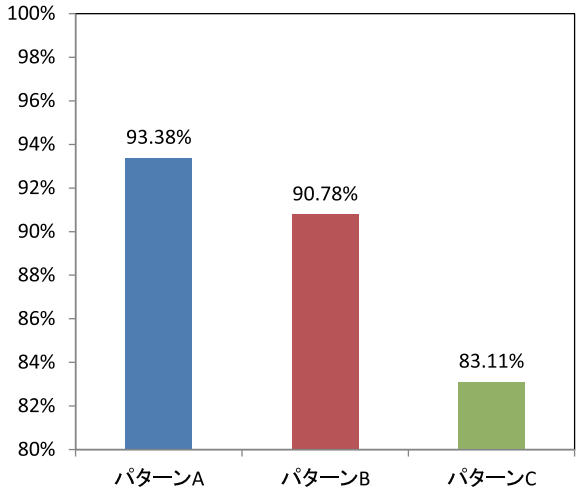

図 5 システムの利用率

す．パターン A の利用率は約 93%でパターン B の利用率は約 91%と比較的近い値であるが，パターン C は利用率が約 83%と他と比べて低くなっている．後述するように，パターン C で実行されたジョブを分析した結果，実行された大規模ノードのジョブが他のパターンに比べて非常に少ないことが判った．これは，パターン C の場合は実行中のジョブはスケジューリング期間のすべてを専有するため，大規模ノードのジョブが実行されにくくなっているのが原因と考えられる．パターン A とパターン B の利用率の差は，12 時間以内のスケジュール区間の細かさによるものである．パターン A とパターン B を比較すると，システム利用率はパターン A が高く，スケジューリングコストはパターン A が低い．これらから，システム利用率とスケジューリングコストの両方でパターン A は優位であることがわかる．

3.3 ジョブの実行待ち時間

図 6 にジョブのノード規模と指定経過時間別の実行までの待ち時間を示す．この図からもパターン C は大規模ノードのジョブが実行されずにスケジューラのキューに積み上げられ，それ以外の小規模なジョブが短い待ち時間で実行されたことがわかる．パターン C はスケジューリング区間が一つのため，短いジョブも長いジョブもそのスケジューリング区間を専有することになる．そのため，ジョブの指定経過時間が異なっても待ち時間がほぼ等しいという結果になったと考えられる．

3.4 実行されたジョブの数

図 7 にシミュレーション期間内に実行されたジョブの数をジョブの規模と指定経過時間別に示す．パターン A,B,C 共にほぼ同じ傾向を示している．パターン C の 101 以上 342 ノード以下で実行したジョブの数は 59 個だったが，パターン A,B はともに 171 個であった．パターン C とそれ以外を比べるとシステムの利用率の差は大規模なジョブが実行されたかどうかに依存していることがわかる．

4. 関連研究

これまでに様々なジョブスケジューラやスケジューリングアルゴリズムの研究開発が行われてきた．大規模環境で代表的なスケジューラとして SLURM[4] が挙げられる．SLURM はオープンソースのスケジューラで，ユーザ自身でスケジューリングアルゴリズムを追加できる．SLURM は Tianhe-2(中国:National Super Computer Center) や Sequoia(Lawrence Livermore National Laboratory)，Piz Daint(Swiss National Supercomputing Centre) など TOP500 リストの TOP10 のうち半数以上で用いられている．「京」では「京」用にカスタマイズされた富士通社製の Parallelnavi が用いられている [5]．Soner[6] らは SLURM に GPU などのヘテロ環境への対応や整数計画法を使ったジョブスケジューリング手法を提案している．他にもジョブスケジューリングアルゴリズムはいくつも提案されているが，それらはより公平性を目指しているものであったり，電力管理を目指しているものであったりする．

提案手法は実際にジョブスケジューラを実装するときに時間をどのような粒度で扱うと良いのかという問題に対し，粒度の粗密でシステムの利用率やジョブの待ち時間およびスケジューリングに要する時間がどのように変化するかを示し，粗密を組み合わせることが良好な結果につながることを示したものである．提案手法は既存のスケジューリングアルゴリズムと排他的なものではなく直交する手法のため，既存手法にも適用することができる．

5. まとめ

本稿では，様々なジョブを効率よくスケジューリングする手法として区間ジョブスケジューリング法を提案した．区間ジョブスケジューリング法は，時間軸方向をいくつかのスケジュール区間に分割し，ジョブの指定経過時間をスケジュール区間単位に切り上げてジョブをスケジューリングする．短いスケジュール区間を用いると分割数が多くなるため，一回のスケジューリングに要するコストは大きくなる．一方，長いスケジュール区間を用いると分割数は少なくなり，スケジューリングに要するコストは小さくなる．提案手法を 3 つのスケジュール区間のパターンで評価した結果，システムの利用率は直近の未来は短いスケジュール区間 (10 分) を用い，遠い未来には長いスケジュール区間 (8 時間) を使った場合に約 93%と一番高くなった．また，スケジュール区間とスケジュール期間が等しい場合，つまりスケジュール区間がひとつの場合のシステムの利用率は約 83%であった．ジョブが投入されてから実行されるまでの待ち時間を評価した結果，スケジュール区間が長いほど

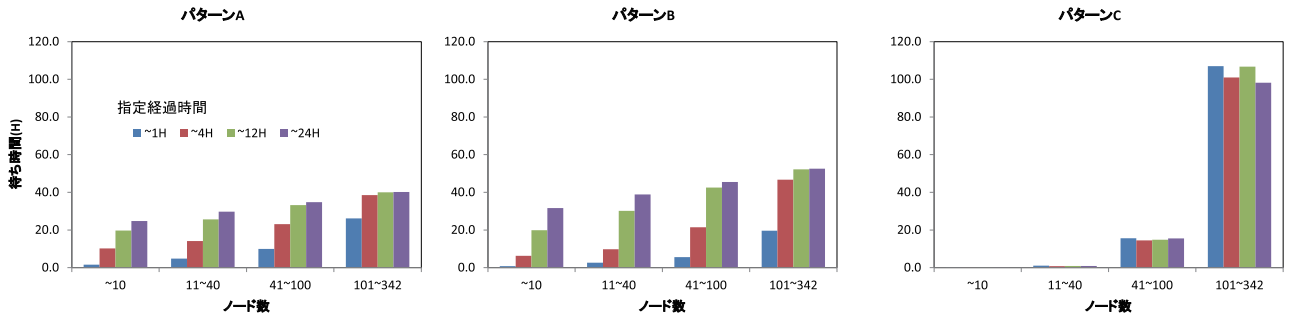

図 6 ジョブの実行待ち時間

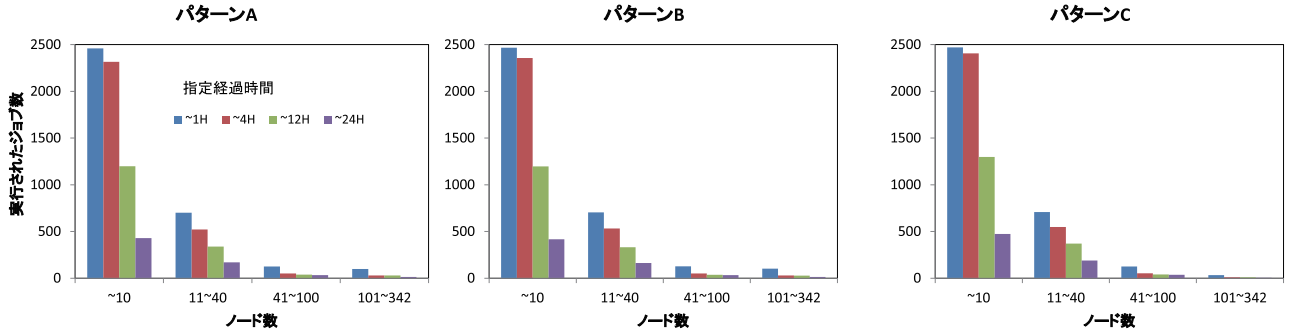

図 7 実行されたジョブの数

ノード数の少ないジョブの実行待ち時間が短くなることがわかった．実行されたジョブ数を調べると，スケジュール区間がひとつパターンでは大規模ジョブの実行数が他のパターンと比べて約 1/3 であり，システム利用率の差は大規模ジョブの実行数の差にあることがわかった．提案手法はスケジュール区間の長さを変化させることで，スケジューリングに要する時間とシステムの利用率を調整することができる．スケジュール区間の最適値はスケジューラの性能やジョブの特性，およびシステムの負荷によって変わると考えられ，それを求める方法は今後の課題である．また，提案手法にファイルステージングを組み合わせた場合についても評価していきたいと考えている．

参考文献

[1] TOP500: TOP500 Supercomputer Sites, Top500.org (online), available from ⟨http://www.top500.org⟩ (accessed 2014-9-1).
[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. and Yokokawa, M.: The K computer Operations: Experiences and Statistics., *International Conference on Computational Science ICCS2014*, pp. 576–585 (2014).
[3] 山本啓二，宇野篤也，塚本俊之，菅田勝文，庄司文由：スーパーコンピュータ「京」の運用状況，情報処理， Vol. 55, No. 8, pp. 786–793 (2014).
[4] Jette, M. A., Yoo, A. B. and Grondona, M.: SLURM: Simple Linux Utility for Resource Management, *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pp. 44–60 (2002).
[5] 平井浩一，井口裕次，宇野篤也，黒川原佳：スーパーコンピュータ「京」の運用管理ソフトウェア，*FUJITSU*, Vol. 63, No. 3, pp. 287–292 (2012).
[6] Soner, S. and Ozturan, C.: Integer Programming Based Heterogeneous CPU-GPU Cluster Scheduler for SLURM Resource Manager, *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication*, HPCC '12, pp. 418–424 (2012).