Vol.2014-HPC-146 No.1
2014/10/2

エネルギー効率を考慮した電力制約下でのスループット指向ジョブスケジューリング手法

黄 巍[1,a)]　岩澤 直弘[1,2]　カオ タン[1,2]　和 遠[1,2]　近藤 正章[1,2]　中村 宏[1]

アブストラクト：HPC データセンタにおいては，消費電力がシステムの設計や性能において最も重要な制約条件となると考えられている．そこで従来から，ある電力制約の下でシステム全体のスループットを最大化するためのジョブスケジューリングに関する研究が行われてきた．しかし，これまでは各ジョブのエネルギー効率についてはあまり考慮はされてこなかった．本稿では，システム全体の電力制約下で使用電力量を考慮しつつ，ジョブ毎の電力制約値及び使用ノード数を最適化することで，各ジョブに対してシステムスループットだけではなくエネルギー効率も同時に考慮したジョブスケジューリング手法の構築を検討する．予め用意された各ジョブの実システム上の性能・電力データをもとにしてスケジューリングを行うシミュレータを作成し実験を行い，従来のスループット重視のスケジューリングに比べ，消費エネルギーを大きく削減できることがわかった．

1. はじめに

現時点で最高性能を持つスーパコンピュータシステムの Tianhe-2 は 33PFLOPS 超の性能を達成しているが，そのために 18MW 近い消費電力を必要とする [1]．2020 年あたりに実現されると予想されるエクサスケール級の HPC システムは，20〜30MW と同程度の電力で現在世界トップクラスのスーパーコンピュータの 30〜50 倍近い性能向上が求められる．そのため，将来の HPC システムでは，消費電力がシステムの設計や実効性能を制約する最大の要因になると考えられており，限られた電力資源を有効利用することは，今後の大規模 HPC システムにおける重要な研究課題の一つである．

これまで，HPC システム上で実行される各並列アプリケーションの電力効率を向上させるための手法や [4]，電力制約下で性能向上を図る手法についての研究が行われてきた [3], [6]．一方で，大規模なスーパーコンピュータシステムでは，同時に複数のジョブが実行されることが通常であり，アプリケーション個々の電力最適化のみならず，システム全体の電力制約を考慮したスケジューリングや電力配分も重要である．そこで，近年では電力制約下でシステムのスループットを最大化するための研究も行われている [2], [7]．

従来の電力制約下での複数ジョブ実行を考慮したスケジューリングに関する研究では，スループットを向上させることのみが目的で，各ジョブのエネルギー効率に関してはあまり考慮されてこなかった．一方で，システムの電力が増大するにつれ，運用にかかる電気料金としてのコストは非常に大きく，HPC データセンタにおいてもジョブ実行に必要なエネルギーも性能同様に考慮すべき重要な指標である．

そこで，本稿では各ジョブのエネルギー効率を考慮しつつ，電力制約下でのシステムのスループットを向上させるためのジョブスケジューリング手法を検討する．具体的には，文献 [2] で提案されているジョブ毎の電力制約値と使用ノード数を最適化することで，電力制約下でのスループットを最大化させる手法をベースとし，エネルギー効率を最適化の際の評価関数に加えることで，性能とエネルギーの両者を考慮したスケジューリング手法を構築することを目指す．文献，および実ベンチマークの実行結果から得られた性能・電力モデルを基に，シミュレーションによる評価を行った結果，構築したスケジューリング手法によりジョブ実行の消費エネルギーを削減できることがわかった．また，システムが性能重視か消費エネルギー重視かに応じて，スケジューリング機能を調節できることがわかった．

2. 関連研究

従来から，ある 1 つの並列ジョブを対象とした電力性能

[1] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technologys, The University of Tokyo
[2] 独立行政法人科学技術振興機構 CREST
CREST, JST
[a)] huang@hal.ipc.i.u-tokyo.ac.jp

最適化に関する研究は多数行われてきた．例えば，与えられた消費電力の上限を超えない範囲で，システム内の各要素間で電力を適切に配分することで，アプリケーションの実効性能を最大化する研究も行われている．文献 [3] では同期ポイントにおける各スレッドや各プロセスの待ち時間を考慮して，各ノード・CPU の動作周波数を設定することで，アプリケーションの性能を低下させずに消費電力を削減する手法が提案されている [4]．近年の Intel 社のプロセッサに備えられているプロセッサ部と DRAM 部の消費電力を観測・制御するためのインタフェースである RAPL (Running Average Power Limit)[9], [10] を用いることで，CPU および DRAM の消費電力測定と制御を行うことが可能である．この機能を利用して，使用ノード数の最適化と CPU・DRAM 間で電力の配分を行う手法も提案されている [5].

近年，HPC 計算分野で広く用いられる GPU では，その消費電力は CPU と比較しても大きいが消費電力あたりの性能は高い．CPU と GPU 間に割り当てる処理量と，それぞれの動作周波数をあわせて変更することで，与えられた消費電力制約下で GPU アプリケーションの性能を最大化するための手法も提案されている [6].

個々のジョブの電力性能最適化ではなく，複数ジョブの電力性能を最適化することを目的に，Etinski らは計算機システムに供給される電力資源を最大限利用するためのスケジューリング手法を提案している [7]．Dynamic Voltage Frequency Scaling (DVFS) を利用して，HPC 計算機システム内で同時に実行されている各ジョブのプロセッサの周波数・電源電圧を変更することで，各ジョブを適切な消費電力で動作させつつ，最大限に電力を使用するものである．ただし，一般的にはメモリアクセス頻度などに依存して，同じ周波数を用いても各ノードの消費電力はジョブ毎に異なる可能性がある．また，当該論文の目的は電力資源の最大限の利用であるため，ジョブの実行速度や消費エネルギーについては十分に考慮されていない．

これに対して，Sarood らは消費電力制約下で，システム全体のスループットを最大化するためのスケジューリング手法を提案した [2]．このスケジューリング手法により，ジョブごとに適切なノード数・電力キャップを設定することができ，システム全体のスループット最大化することができる．

3. 電力制約下でのジョブスケジューリング

本章では，従来の電力制約下でのジョブスケジューリング手法として文献 [2] で提案されている手法について述べる．

3.1 ジョブスケジューリング手法の概要

文献 [2] では，ある計算機センターのシステムが利用可能な総消費電力の制約が与えられた際に，システム全体のスループットを最大化するためのスケジューリング手法である *Power Aware Resource Manager (PARM)* を提案している．PARM の概要を図 1 に示す．PARM はイベント発生時にどのジョブをどのような設定で実行するかを決定するスケジューラ (*scheduler*) と，実際に与えられた設定で指定されたジョブを実行させる実行フレームワーク (*Execution framework*) で構成される．スケジューラは新たなジョブがキューにサブミットされた際といずれかのジョブの実行が完了またはエラーなどの原因で終了した際をトリガーとして起動される．スケジューラは既に実行中の全ジョブ，およびキューで待ち状態のジョブの中から新たに実行するジョブを選択し，各ジョブの実行設定として利用ノード数と使用電力の上限 (電力キャップ) を決定する．なお，あるジョブが一旦実行を開始すると，そのジョブは完了まで止めることはしないと仮定する．キューの中からどのジョブを選択するか，および各ジョブにどのような実行設定を与えるかには多数の候補があり得るが，組み合わせ最適化問題を解くことで計算機センターのスループットを最大化する最適解を求めることが PARM の役割である．

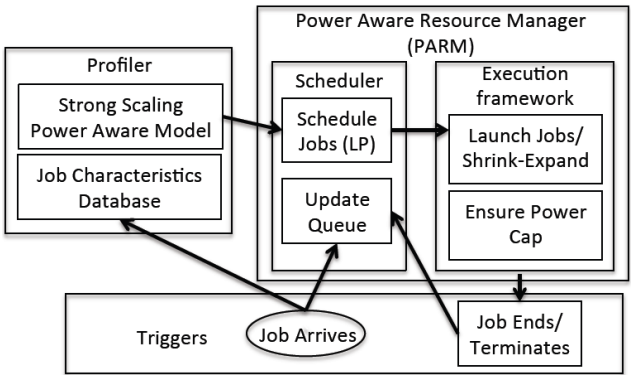

図 1 Power Aware Resource Manager の概要 (出展 [2])

一方で，実行フレームワークはスケジューラから与えられた設定を用い，各ジョブを指定ノード数と指定電力キャップ値で実行させる．電力キャッピングの実現には，Intel 社のプロセッサに備えられている RAPL を利用することを想定する．通常，実行ノード数を動的に変更することは，大規模並列処理を行うプログラムとして一般的に利用されている並列プログラミングインタフェースの MPI を用いた場合には簡単ではない．そこで，文献 [2] では，キュー内のジョブがシステム上で実行される際に，ある範囲からノード数を選択して実行することが可能なケース (Moldable ジョブ)，および Charm++[11], [12] などの runtime を用いることで実行時にも使用ノード数を変更可能なケース (Malleable ジョブ) を主に仮定している．

3.2 最適化問題としての定式化

前述のように，スケジューラはトリガーイベントの発生の度に組み合わせ最適化問題を解く必要がある．文献 [2] で用いられている整数線形計画問題としての定式化を式 (3.1)-(3.5) に示す．なお，当該問題への入力は実行中またはキュー内で実行待ちのジョブの集合 $\mathcal{J}$，あるノード数 n とある電力制約 p で実行した際の各ジョブ $j \in \mathcal{J}$ の実行時間であり，出力は実行すべきジョブとその実行設定 (n, p) である．また，表 1 に式中で利用する記号の意味を示す．

目的関数

$$\sum_{j\in\mathcal{J}} \sum_{p\in\mathcal{P}_j} \sum_{n\in\mathcal{N}_j} s_{j,n,p} * w_{j,n,p} * x_{j,n,p} \tag{3.1}$$

2 値変数 x の満たす条件

$$\sum_{p\in\mathcal{P}_j} \sum_{n\in\mathcal{N}_j} x_{j,n,p} \leq 1 \qquad j \in I \tag{3.2}$$

$$\sum_{p\in\mathcal{P}_j} \sum_{n\in\mathcal{N}_j} x_{j,n,p} = 1 \qquad j \in \mathcal{I} \tag{3.3}$$

制約条件

$$\sum_{j\in\mathcal{J}} \sum_{p\in\mathcal{P}_j} \sum_{n\in\mathcal{N}_j} n * x_{j,n,p} \leq N \tag{3.4}$$

$$\sum_{j\in\mathcal{J}} \sum_{p\in\mathcal{P}_j} \sum_{n\in\mathcal{N}_j} (n(p + W_{base}))x_{j,n,p} \leq W_{max} \tag{3.5}$$

表 1 定式化で用いる記号

記号	説明
N	ノードの総数
J	すべてのジョブの集合
$\mathcal{I}$	現在実行中のジョブの集合
I	キュー内にある実行待ちジョブの集合
$\mathcal{J}$	サブミットされたが，実行し終わってないジョブの集合
N_j	ジョブ j が実行できるノード数の集合
P_j	ジョブ j が実行できる電力設定の集合
w_j	ジョブの優先度付けのための重み
α	w_j の中で，優先度と性能を調節するための変数
$x_{j,n,p}$	ジョブ j が n ノード上で電力 p で実行する場合は 1，その他の場合は 0
t_{now}	現在の時刻
t^a_j	ジョブ j がサブミットされた時刻
W_{base}	マシンのベース電力
$t_{j,n,p}$	ジョブ j をノード数 n，電力 p で実行した場合の実行時間
$s_{j,n,p}$	ジョブ j をノード数 n，電力 p で実行した場合の最小構成に対するスピードアップ

本定式化においては，ジョブ j をノード数 n，電力 p で実行した場合の最小構成に対するスピードアップを見積もる必要がある．この組み合わせは非常に膨大な数になるため，いかに効率よく見積れるかが重要である．そこで，文献 [2] では *Strong Scaling Performance Model* という性能モデルを導入している．

まず，ジョブ j のスピードアップを式 (3.6) のように定義する．

$$s_{j,n,p} = \frac{t_{j,min(N_j),min(N_j),min(P_j))}}{t_{j,n,p}} \tag{3.6}$$

ここで，$s_{j,n,p}$ はジョブ j をノード数 n，電力 p で実行した場合の実行時間と，ジョブ j が実行可能な最小ノード数 $min(N_j)$ と最小電力 $min(P_j)$ で実行した際の比である．式 (3.1) で示した目的関数は，全実行ジョブのスピードアップ $s_{j,n,p}$ を足し合わせたものである．そのため，目的関数を最大化することは，実行ジョブ全体を高速に実行するのに適するノード数と電力キャップ値を各ジョブに対して選ぶことに相当する．$t_{j,n,p}$ の性能モデルの詳細に関しては，文献 [2] を参照されたい．なお，4.2 節では，文献 [2] の性能モデルを本稿のために改変したモデルについて述べる．

目的関数として $s_{j,n,p}$ のみを使用すると，大規模なジョブを実行するよりも複数の小さなジョブを実行する場合の方が目的関数の値が大きくなり，大規模ジョブが実行され難くなる傾向がある．その結果，大規模ジョブの待ち時間が長くなり，ジョブ間の公平性が悪化する．この対処のために，キューで長く待たされているジョブの優先度を高くするべく，以下のように重み w_j を定義する．

$$w_j = (t^{rem}_{j,min(N_j),min(P_j)} + (t_{now} - t^a_j))^\alpha \tag{3.7}$$

ここで，$t^{rem}_{j,min(N_j),min(P_j)}$ は，ジョブ j を最小ノード数 $min(N_j)$，最小電力 $min(P_j)$ で現時刻からジョブ完了までかかる残り時間の見積もり値である．これをジョブ j がサブミットされた時間を足し合わせたものをジョブの重要度とすることで，より実行時間の長いジョブ，あるいは待ち時間の長いジョブの優先度が高くなる．なお，α は 0 から 1 までの値域を持つ変数であり，0 に設定した場合は性能重視で公平性を無視したスケジューリング，1 に設定した場合は公平性を重視し性能低下を許容するスケジューリングとなる．

3.3 従来のジョブスケジューリング手法の課題

前節で述べた最適化の目的関数として，スピードアップを表す $s_{j,n,p}$ のみが存在することからもわかるように．従来の電力制約下でのジョブスケジューリング手法は，スループットを向上させることのみが目的であり，各ジョブのエネルギー効率に関しては考慮されていない．一方で，計算機センターで消費される計算機システムの電力が増大するにつれ，運用にかかる電気料金コストは非常に大きく，ジョブ実行に必要な消費エネルギーも性能と同様に考慮すべき重要な指標である．例えば，スループット最大化のために，高いプロセッサのクロック周波数を用いて動作させるよりも，低い周波数と電源電圧を用い，消費電力を抑えて実行する方が一般的にエネルギー効率は高くなる．そのため，エネルギー効率も考慮した場合には，従来のス

表 2　エネルギーに関する用語

記号	説明
$Er_{j,n,p}$	消費エネルギーの削減度合い
$E_{j,n,p}$	ジョブ j をノード数 n，電力 p で実行した場合の消費エネルギー
β	性能と消費エネルギーの重要度の重み

ケジューリングとは異なるジョブを異なる実行設定で実行させる場合が最適となる可能性がある．

4. エネルギーを考慮した電力制約下でのジョブスケジューリング

本稿では，従来のスケジューリング手法の課題をふまえ，PARM スケジューリング手法の拡張を検討する．一般的に，性能とエネルギー効率にはトレードオフの関係がある．計算機センターの運用ポリシーやエネルギーコストの違いにより，性能とエネルギー効率のどちらを重視するかは運用ポリシーに依存すると考えられる．そこで，性能と消費エネルギー効率のどちらをどれだけ重要視するかを重みとして与えられるようにしつつ，両指標を考慮したスケジューリング手法の構築を目指す．

4.1 目的関数の拡張

3.2 節で述べた PARM 定式化の拡張として，消費エネルギーに関する項目を目的関数に含めることを考える．拡張した目的関数を式 (4.1) に示す．また，ここで新たに使用する記号に関する説明を表 2 に示す．

$$\sum_{j\in\mathcal{J}}\sum_{p\in\mathcal{P}_j}\sum_{n\in\mathcal{N}_j}(\beta * s_{j,n,p} * w_{j,n,p} + (1-\beta) * Er_{j,n,p})x_{j,n,p} \tag{4.1}$$

ここで $Er_{j,n,p}$ はジョブ j を設定 (n,p) で，すなわちノード数 n，電力 p で実行した場合に対する，最小ノード数と最小電力で実行した場合のエネルギー削減率であり，以下のように定義できる．

$$\begin{aligned} Er_{j,n,p} &= \frac{E_{j,min(N_j),min(P_j)}}{E_{j,n,p}} \\ &= \frac{t_{j,min(N_j),min(P_j)} * min(N_j) * (min(P_j) + W_{base})}{t_{j,n,p} * n * (p + W_{base})} \\ &= s_{j,n,p} * \frac{min(N_j) * (min(P_j) + W_{base})}{n * (p + W_{base})} \end{aligned} \tag{4.2}$$

(n,p) で実行した場合の消費エネルギーが少ないほど $Er_{j,n,p}$ は大きくなることから，スピードアップ $s_{j,n,p}$ が大きいほど，また $Er_{j,n,p}$ が大きいほど目的関数の値が大きくなり，それを最大化するように最適化することで，エネルギーを考慮したスケジューリングを行う事ができる．

なお，式 (4.1) にある β は性能と消費エネルギー効率を調整する重みである．β を設定することで，実行時間重視か消費エネルギー効率重視かを変更することができる．β が 0 の場合では，従来研究と同じ目的関数となり，完全に性能を重視したスケジューリングとなる．一方，β が 1 の場合は，完全に消費エネルギー効率を重視したスケジューリングとなる．本稿では，β がジョブに対して個別に決定されるのではなく，一律同じ値を用いる．

4.2 スピードアップモデル

本節では，目的関数にあるスピードアップを見積もるための性能モデルについて説明する．文献 [2] でも性能モデルが提案されているが，より簡易にモデルを構築・評価できるように，本稿では一部性能モデルを変更する．

ここでのスピードアップモデルは，以下のようにノード数・実行性能モデルと電力・実行性能モデルの二つに分けて構成される．

$$s_{j,n,p} = s_j(n) * s_j(p) \tag{4.3}$$

以下，それぞれについて詳述する．

4.2.1 ノード数・実行性能モデル

あるジョブは，平均並列度 A と並列度の分散 σ を用いて，使用ノード数のを変更させた場合の実行時間 $t(n)$ を次のように表現できる [8]．

$$t(n) = \begin{cases} \dfrac{T_1 - \frac{T_1\sigma}{2A}}{n} + \dfrac{T_1\sigma}{2A} & n < A \quad \text{(4.4a)} \\ \dfrac{\sigma(T_1 - \frac{T_1}{2A})}{n} + \dfrac{T_1}{A} - \dfrac{T_1\sigma}{2A} & A \le n \le 2A-1 \quad \text{(4.4b)} \\ \dfrac{T_1}{A} & n \ge 2A-1 \quad \text{(4.4c)} \end{cases}$$

ここで，n はノード数であり，T_1 はジョブが単一ノードで実行される場合の実行時間である．式（4.4a）では，ノード数が A より小さい場合には，ノード数に対するのスケーラビリティが高いことを示しおり，式（4.4b）では，ノード数が A より大きい場合にはスケーラビリティが大幅に減少することを意味している．さらに，式（4.4c）にあるように，n が $2A-1$ よりも大きい場合は実行時間が T_1/A のままとなり，ノード数を追加しても実行時間の減少は得られないことを示している．このモデルにより，ジョブの特徴として A, σ, T_1 を求めることができれば，任意のジョブに対してノード数を変化させた際の性能を見積もることが可能となる．ここで，スピードアップ $s_j(n)$ は $t(n)/T(1)$ として表すことができる．

なお，次章の評価では実行するジョブの A，σ の値は文献 [8] に記載されている値を用いる．

4.2.2 電力・実行性能モデル

本研究では RAPL を用いて，CPU パッケージの電力ドメインに対して実際に電力キャッピングの値を設定し，実際にジョブを実行して実行時間を測定することで，電力・

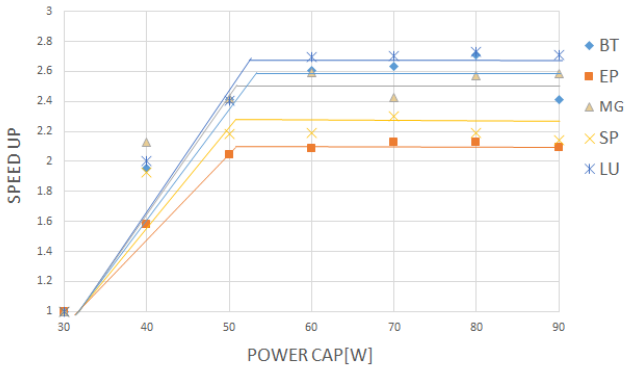

図 2　16 コアでの各ジョブの電力・実行速度モデル

実行性能モデルを構築する．

図 2 に Nas Parallel Benchmarks (NPB) 中の 5 種類のプログラム (BT, MG, SP, LU, EP) を選択し，1 ノード 16 コアのシステムを用い電力キャップ値を 30W から 90W に変化させた場合のスピードアップを示す．なお，30W 制約時の実行時間を規準とした場合のスピードアップである．図より，電力キャップ値を大きくしていくと，5 種類のプログラムとも 50W 電力キャップまでは性能向上が見られるが，50W より高いキャップ値では，速度向上が見られない．また，30W から 50W の間では，電力キャップ値に対してほぼ線形にスピードアップすることがわかる．本研究では，次章の評価にこの 5 種類のジョブを用い，電力・実行性能モデル $s(j,p)$ を次のように定義する．

$$s_j(p) = \begin{cases} (p-30) * m_j + 1 & p < 50 \quad (4.5a) \\ s_j(50) & 50 \leq p \quad (4.5b) \end{cases}$$

ここで，m_j はジョブ j の電力キャップの値が 30W から 50W の間の近似線の傾きである．

5. 評価実験

本稿で検討したエネルギー効率を考慮した電力制約下でスケジューリング手法の効果を調べるために，前節で示したモデルとスピードアップに関するパラメータを利用してシミュレーション行い評価する．評価では，前節で述べた NPB の 5 種類のプログラムを用い，その中からジョブセットをランダムに生成する．また，目的関数のパラメータである β を変化させなからシミュレーション評価を行い，その結果を単純に FIFO ポリシーでスケジューリングした場合，および文献 [2] の従来手法と比較する．

5.1 実験環境

本稿ではシミュレーション環境として Matlab を用いた．また，組み合わせ最適化問題の解法に整数線形計画法 (従来手法)，および非線形整数計画法 (本稿での拡張手法) を用いると探索に時間がかかり解が得られないことがあったため，今回は電力キャップ値と利用ノード数の候補をある程度限定し (ノード数候補は 1, 2, 4, 8, 16, 32 の 6 通り，電力キャップ値は 30W, 34W, 38W, 42W, 46W, 50W の 6 段階)，全探索を行うことで解を求めた．

ジョブセットは BT, MG, SP, LU, EP の 5 種類のプログラからからランダムに 20 個を生成して評価する．表 3 は実験に用いた各プログラムのパラメータである．これらは，Intel Xeon　CPU E5-2670-v2 2.50GH を 2 ソケット搭載する実機により得られたデータ，および文献 [8] に記載されたデータを基に作成した．また，ジョブがサブミットされるタイミングは λ=10 のポアソン分布に従うと仮定した．NPB は実際には実行時に仕様ノード数を変更することはできないが，本実験ではスケジューリングの効果を検証するために，各ジョブが実行時に使用ノード数を変更可能な Malleable ジョブを仮定して評価を行った．

表 3　プログラムのパラメータ ($min(P)$=30W, $min(N)$=1)

ジョブ名	$t_{min(P),min(N)}$	A	σ	m
BT	291.747[s]	7	0.02	0.055
MG	22.473[s]	18	0.04	0.070
SP	297.194[s]	20	0.05	0.080
LU	228.374[s]	18	0.01	0.075
EP	43.867[s]	Inf	0	0.065

評価対象 HPC 計算機システムの仮定として，16 コアの CPU を持つ計算機を 1 ノードとし，最大ノード数は 32 ノードとした．なお，CPU ソケット毎に常に消費されるベース電力を 10W と仮定し，本評価では CPU の電力のみを対象に評価を行う．この際に，最大許容電力は 800W に設定して実験を行った．

5.2 評価結果

比較評価のための FIFO スケジューリングには各ジョブの実行設定 (n,p) を最適化する枠組みはないため，ノード数 n と電力キャップ p を固定して評価を行う．この際に，各ジョブを最小ノード数 $min(N_j)$ と最小電力 $min(P_j)$ で実行する場合 ($FIFO_{min}$)，最大電力 $max(P_j)$ で実行できるようノード数を設定した場合 ($FIFO_{max}$)，最大電力と最小電力の中間の電力で実行できるようノード数を設定した場合 ($FIFO_{mid}$) のそれぞれを評価した．

図 3 に，作成したジョブセットに対して $FIFO_{max}$, $FIFO_{mid}$, $FIFO_{min}$，およびスループットを重視した従来のスケジューリング，本稿で拡張したエネルギーを重視したスケジューリングのそれぞれにおける総実行時間と総消費エネルギーを示す．ここで，総実行時間とは，最初のジョブがキューにサブミットされてから最後のジョブの実行が完了するまでの時間である．

図より FIFO では単純にジョブのサブミット順に実行され，またノード数と電力キャップ値も固定の値が使われることから実行時間が最適化したものに比べて長く，また消

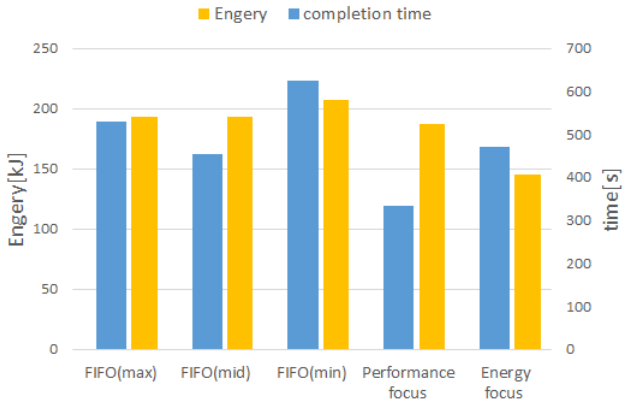

図 3　総実行時間と総消費エネルギーの比較

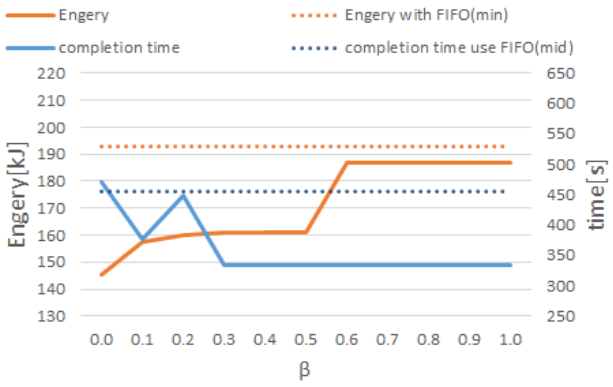

図 4　β を変化させた場合の結果

費エネルギーも大きい．$FIFO_{max}$ よりも $FIFO_{mid}$ の実行時間が短いのは，$FIFO_{max}$ は平均並列度 A を超えても多くのノードを利用して実行するためにシステム全体の効率が悪くなったための考えられる．実行時間が長くなるとベース電力があるために消費エネルギーも増加する．

一方で，性能重視のスケジューリング (式 (3.1) の目的関数を使用) では，FIFO の中では最も性能が良い $FIFO_{mid}$ に対しても，総実行時間が 26%も減少していることがわかる．また実行時間が短くなったことで，総消費エネルギーも FIFO に比べてわずかに削減できている．さらに，本稿で検討したエネルギーを考慮したスケジューリングでエネルギー重視の戦略をとった場合 (β=0.0 に設定) には，性能重視のスケジューリングに比べて総実行時間は長くなっているものの，約 22%もの消費エネルギーが削減されており，実際に消費エネルギー削減効果が大きいことがわかる．また，総実行時間も $FIFO_{mid}$ と比べてもわずかの増加にとどまっている．この結果より，本稿で拡張したスケジューリングが，計算機センターにとって大きなコストになる消費エネルギーを削減には効果的であると結論付けることができる．

図 4 は先の実験と同一のジョブセットを用い，β の値を 0.0〜1.0 の間で 0.1 刻みで変化させながらシミュレーション実験をした結果である．同じく，総実行時間と総消費エネルギーを示している．参考に，FIFO で最も性能の良かった $FIFO_{mid}$ の結果も点線で示している．なお，β=1.0 の場合は従来研究のスケジューリング手法となり，β の値が小さいほど消費エネルギー重視のスケジューリングとなる．β=0.0 の場合が図 3 のエネルギー重視スケジューリングに相当する．

図 4 より，β の値が小さくなるにつれ消費エネルギー重視のスケジューリングになることから，消費エネルギーが減少している．一方で，総実行時間は長くなる傾向にある．一部，β=0.2 の場合に総実行時間が他に比べて長くなっているが，この理由は検討中である．このように，β のパラメータ値を変更することで，各計算機システムの運用に合わせて性能と消費エネルギーを調整することができる．これも本稿で拡張したスケジューリング手法の利点になると考えられる．

6. おわりに

本稿では，性能とエネルギー効率の両方を考慮した，HPC 計算機システム向けの電力制約下でのスケジューリング手法について検討し，性能のみを重視した従来手法と比較しつつ評価を行った．従来研究で用いられていた最適化問題の目的関数にエネルギー効率の項を追加して再定義することで，消費エネルギーも考慮したスケジューリングを行うことが可能になる．シミュレーションにより評価実験を行い，従来手法に比べて消費エネルギーを削減できることがわかった．また，目的関数内の重み変数を変化させることで，スケジューリングにおいて性能またはエネルギー効率の重視度を変化させることが可能であると分かった．今後の課題として，組み合わせ最適化問題の解法を工夫し，より多くの探索候補の中から最適な実行設定を選択すること，またより多数のジョブ数で評価をすることなどが挙げられる．

謝辞　本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST) の研究プロジェクト「ポストペタスケールシステムのための電力マネージメントフレームワークの開発」の助成により行われたものである．

参考文献

[1] Top 500 Supercomputer Sites: http://www.top500.org/.
[2] O. Sarood, A. Langer, A. Gupta, and L.V. Kale: Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget, Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'14), (2014).
[3] 會田　翔，三輪　忍，中村　宏：ロードバランスを考慮した電力制約下における CPU の DVFS 制御，情報処理学会研究報告ハイパフォーマンスコンピューティング (HPC), Vol.2014-HPC-143, No.23, pp. 1–8, (2014).
[4] O. Sarood, A. Langer, L. Kale, B. Rountree, and B. de Supinski: Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems, Proc. of 2013 IEEE International Conference on Cluster

Computing, pp. 1–8 , (2013).
[5] 稲富雄一, 吉田匡兵, 深沢圭一郎, 上田将嗣, 青柳　睦, 井上弘士：電力指向型次世代スーパーコンピュータを想定した HPC アプリケーションの性能最適化～量子化学計算の場合～, 情報処理学会研究報告ハイパフォーマンスコンピューティング (HPC), Vol.2013-HPC-142, No.30, pp. 1–6, (2013).
[6] T. Komoda, S, Hayashi, T. Nakada, S. Miwa, and H. Nakamura: Power Capping of CPU-GPU Heterogeneous Systems Through Coordinating DVFS and Task Mapping, Proc. of 2013 IEEE 31st International Conference on Computer Design, pp. 349–356, (2013).
[7] M. Etinski, J. Corbalan, J. Labarta, M. Valero: Parallel job scheduling for power constrained HPC systems, Parallel Computing, Volume 38, Issue 12, pp. 615-630, (2012).
[8] A. B. Downey: A Model for Speedup of Parallel Programs, (1997).
[9] Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A. and Weissmann, E.: Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge, *IEEE Micro*, Vol.32, No.2, pp. 20–27, (2012).
[10] David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R. and Le, C.: RAPL: Memory Power Estimation and Capping, *Proceedings of the 16th ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pp. 189–194, (2010).
[11] L. Kale: The Chare Kernel Parallel Programming Language and System, Proc. of the International Conference on Parallel Processing, vol. II, pp. 17-25, (1990)
[12] L. Kale, A. Arya, N. Jain, A. Langer, J. Lifflander, H. Menon, X. Ni, Y. Sun, E. Totoni, R. Venkataraman, and L. Wesolowski: Migratable Objects + Active Messages + Adaptive Runtime = Productivity + performance a Submission to 2012 HPC Class II Challenge,” Parallel Programming Laboratory, Tech. Rep. 12–47, (2012).