情報処理学会論文誌　コンピューティングシステム　Vol.6　No.4　72–82 (Oct. 2013)

電力を考慮したアプリケーション構築のための計算機システムの提案

横山 大作[1,a)]　喜連川 優[1,2,b)]

受付日 2013年4月9日, 採録日 2013年6月26日

概要：ユーザからのサービス需要が時間とともに大きく変動するアプリケーションに対し，オンデマンドに使用リソースを変動させてサービス提供を行えるクラウドコンピューティングと呼ばれる計算環境が広く利用されている．近年では，リソース利用コストの削減に加え，消費電力量の削減が重要な要請になりつつあるが，電源管理において複数のインタフェース規格を使い分けなければならないこと，また計算機，通信機器，記憶装置などの複数のコンポーネントの電源を連携した状態で制御する必要があることなどに，課題があると考えられる．これらの問題点をふまえ，我々は，計算機クラスタ上でユーザプログラムが電源管理を容易に行える統合的インタフェースを提供し，必要なリソースのみ電源を供給することで省電力化を図ることが可能なシステムを構築した．大規模な実クラスタ上で取得されたタスクトレースを利用したシミュレーションを行ったところ，サービスレベルを保ちつつ 13%程度の電力量削減が可能であるとの結論を得た．また，電力制御を加えた探索アプリケーションを実クラスタ上で動作させたところ，総消費電力量を 14%から 23%程度削減できることを確認した．

キーワード：クラウドコンピューティング，エラスティックアプリケーション，省電力，電源管理システム

A Cluster Computing System for Constructing Energy Aware Applications

Daisaku Yokoyama[1,a)]　Masaru Kitsuregawa[1,2,b)]

Received: April 9, 2013, Accepted: June 26, 2013

Abstract: Cloud computing environments that provide on-demand resource allocation features are widely used to serve applications with time-varying demands. On such applications, reducing electric energy consumption is highly required in these days as well as reducing resource consumption. Clusters are often built by several components that are controlled by the several independent power APIs that should be separately managed by application developers. However, these components, such as computing nodes, interconnects, and storages, depend on each others to execute required jobs, and should be controlled in a coordinated way. That makes it difficult to construct energy aware applications. We propose a power controlling interface on cluster environments that provides a simple and unified way to reduce energy consumption through shutting down components that are not required by running jobs. Through a simulation using a large scale task-trace of a real cluster, we show that our approach can reduce energy consumption by around 13%. We also examine an elastic tree-search application on our system, and show that we can reduce energy by between 14% and 23%.

Keywords: cloud computing, Elastic application, energy saving, power controlling system

[1] 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo, Meguro, Tokyo 153–8505, Japan
[2] 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] yokoyama@tkl.iis.u-tokyo.ac.jp
[b)] kitsure@tkl.iis.u-tokyo.ac.jp

1. はじめに

1.1 背景

多数のユーザが利用するウェブサービス，センシングなどのアプリケーションにおいては，昼と夜など時間によってユーザ数が変化したり，大きなスポーツイベント，事故，

天候変化などのイベントにともなって突発的に需要が発生したりするなどの要因で，サービス需要が大きく変動することがしばしば発生する．固定的なサービスインフラを利用していると，需要過多のときにはサービス品質が落ち，需要過小のときには無駄なリソース維持コストがかかるということで，このような需要変化を効率良く処理することができない．この問題に対し，オンデマンドに使用リソースを変動させてサービス提供を行えるクラウドコンピューティングと呼ばれる計算環境が，効率の良い実行環境として着目されている [1]．近年では Amazon EC2 をはじめとする実用的クラウド環境も充実しつつあり，広く利用されている．

クラウドコンピューティングでは，需要に合わせてオンデマンドにリソースを利用し，レスポンスタイムなどのサービスレベルを一定以上に保ちつつ使用リソースの利用コストを最小にすることが求められるが，近年では使用する電力量を削減することも重要な要求になりつつある．電力量はコストを決定する主たる要素になっており，また社会的情勢からも省電力が求められているためである．計算における電力の最適化には，リソースの需要に従って電源を管理し，必要な部分のみが電力を使用するように制御する必要がある．Barroso らは “Energy Proportional” という概念を提案している [2]．これは，電力の消費量を必要な計算量に比例する程度に抑えることが望ましいという考え方である．CPU，ディスクなど計算機リソースの各コンポーネントはそれぞれ，アイドル時の電圧制御 [3] などの様々な技術により省電力化を実現している．しかし，プログラム処理系，スケジューラなどのシステムがこれらの機能を積極的に利用する段階には至っていない．

1.2　実例

例として，我々が利用している計算機クラスタにおいて計算を行うときの消費電力量を測定した結果を示す．計算機クラスタはサーバがトップオブラックスイッチに接続され，それがセンタースイッチで結合される構成をとっている．計算サーバ数は 127，トップオブラックスイッチ（10G）数は 7，センタースイッチ（10G）数は 1，管理用スイッチ（1G）数は 4 である．機器の詳細は**表 1** に示す．

本クラスタは Power Distribution Unit（PDU）として，消費電力測定機能のある Raritan DPXS12-30L-J を利用している．構成機器それぞれについて消費電力を測定した結果を**表 2** に示す．いずれも，20 秒ごとに 3 分間以上計測を行ったときの平均値を示している．サーバについては CPU 負荷中心のアプリケーション（4.2 節の評価実験で利用したコンピュータ将棋プレイヤ）の実行時（busy）と，OS 以外にアプリケーションが動作していないとき（idle）について計測を行った．また，サーバの起動時（boot），終了時（shutdown），停止時（off）の電力についても計測を行った．起動指示時点から 3 分間を起動時，終了指示時点から 1 分間を終了時とし，同一構成である 10 台のサーバでの実験結果の平均値を求めた．サーバの停止時にもごく限られた量の電力を消費していることが分かるが，これは電源の遠隔管理などを担当する管理モジュールによるものである．

表 1　構成機器
Table 1　Computation components.

計算サーバ	DELL PowerEdge R610
CPU	Xeon E5530 × 2（total 8 core）
Memory	24 GB
Disk	500 GB SATA × 4

ネットワークスイッチ	
計算用（10G, 24 port）	Summit X650-24x
管理用（1G, 48 port）	DELL PowerConnect5448

表 2　機器の消費電力
Table 2　Power consumption of each component.

機器	消費電力（watt）
サーバ（busy）	242
サーバ（idle）	115
サーバ（boot）	139
サーバ（shutdown）	119
サーバ（off）	5.8
10G switch	230
1G switch	48

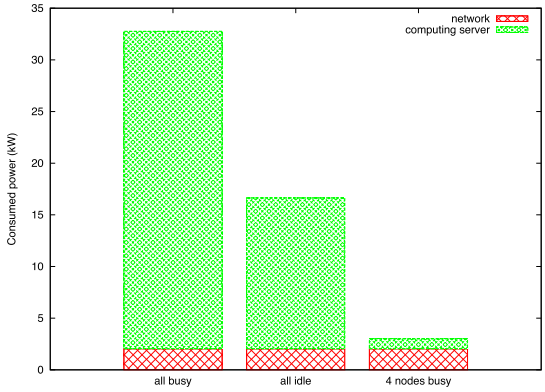

図 1　消費電力想定
Fig. 1　Estimated power consumption.

これらの測定値を用いて，このクラスタを最大限利用しているとき（all busy）と，まったく計算を行っていないとき（all idle）とを考えると，それぞれの消費電力は**図 1** のように求められる．計算サーバに何も仕事がない場合，サーバ内部の省電力機構によってクラスタ全体の電力は半分程度に抑えられていることが分かるが，今仮に 4 台の計算サーバのみを残し，他の計算サーバの電源を落として，4 台のみが busy 状態にある場合（4 nodes busy）を考えると，サーバの停止によって大幅な電力削減が可能であるこ

とが分かる．

また，ネットワーク機器がそれなりの割合の電力を利用していることも読み取れる．特に，4 台のみ実行時にはネットワーク機器の消費電力が計算サーバの倍程度を占めるまでに至っており，電力削減のボトルネックになっていることが分かる．これは，データセンタはネットワーク設備も Energy proportional にすべきであるという Abts らの研究 [4] で述べられたものと同様の結果である．つまり，低負荷時の電力削減を追求するためには，サーバと同時にネットワーク機器についても電源制御を行うことが必要であり，計算機環境全体の状態を把握して破綻なく各機器の電源管理を行えるようなシステムが必要になると予想される．

1.3　提案

多数のアプリケーションが動作するクラウド環境で電力制御を行うためには，現在のところ以下のような点に問題があると考える．

- 機器ごとに様々な電源管理インタフェースが混在しており，取扱いが煩雑である．
- サーバとネットワーク機器など，複数の機器の機能上の関連を考慮して，破綻なく電源制御を行わなければならない．
- どのように機器を制御することが省電力に結び付くのか，適切な電力制御アルゴリズムを構築する必要がある．

我々は，この問題に取り組むために，サービスに必要な最低限度のもの以外の機器を自動的に停止し，消費電力量の削減を図るようなシステムを提案する．ユーザは，必要なリソースを指定して計算環境を利用する．一定時間ユーザが必要としなくなったリソースは，システムによって電源を停止させられる．またこのとき，リソースの依存関係も考慮して最大限の機器を停止することを目指す．

我々は，本提案手法に基づいた電源管理システムをクラスタ上に作成した．また，ユーザが簡単に利用できる統一的なインタフェースを提案し，長期にわたり運用を行っている．この経験から，ユーザが提案インタフェースを利用することは容易であり，プログラム処理系などからの連携が期待できると考えている．

電力 × 経過時間で与えられる電力量を減らすことが省エネルギーのために要請されていると考えたとき，一定時間後に電源を切るというこの電源管理手法によりどの程度の省エネルギー化が可能なのか，その有効性を確認する必要がある．このため，Google が提供する大規模なクラスタでのタスク実行ログを利用した消費電力シミュレーションを行ったところ，提案手法によってサービスレベルを保ったままで消費電力量を 13%程度削減することが可能であることが確認された．また，対象とするプログラム処理系の 1 つとしてワークフローを実行できる GXP を題材に，本電源管理システムと連携するための実装を行った．この処理系上で探索を行う実アプリケーションを用いた実験を行い，Elastic な計算によって実計算環境においても消費電力量が 14%から 23%程度削減できることを確認した．

2.　電源管理システム

2.1　要求

クラスタ環境において，アプリケーションの動作に影響を与えない範囲で適切な電力制御を行う取り組みは多数行われている．たとえば，クラスタに投入されるタスクを処理するワークフローエンジンが電力を削減できるタスクスケジューリングを行う研究 [5] や，サーバの電力制御を活用した Virtual Machine のスケジューリングの研究 [6] など，様々な研究がなされている．また，VMWare 社は，Virtual Machine 実行環境が自動的に VM を集約し，稼働する VM がなくなった計算機を停止する VMWare Distributed Power Management という機能を商用化している．これらは，アプリケーションはブラックボックスとして扱い，アプリケーションを外部から観察した結果をもとにクラスタ環境を制御するものである．

一方，クラウド環境が広く利用されるようになり，オンデマンドな計算資源確保が可能になるにつれて，変動する需要に合わせて利用計算資源を増減させて SLA を守りつつ計算を実行する，Elastic Computing という考え方が重視され始めている．multi-tier ウェブアプリケーション，Key-Value ストア，メッセージキューなど，様々なプログラミングフレームワークやプログラム基盤において，Elastic な計算を実現するための機能が提供されている．たとえば Key-Value ストアの処理系の 1 つである mongoDB では，データを保持する計算ノードをユーザからの指示により増減する機能を提供しており，ユーザは自分のプログラムから，アプリケーションの都合に合わせて適切にノード数やデータの配置を制御することが可能である．

Elastic な計算は，需要の変化と守るべきサービスレベル，およびそのサービスを提供できるリソース構成を理解していなければ実現できないものであり，アプリケーションの側で制御を行わないと実現は難しい．よって，このような計算を行いつつ省電力化を図るためには，アプリケーションの側，プログラム処理系の側からも計算リソースの電力制御を行いたいという要求が生じる．本研究は，このような要求に応える電源管理システムを構築することを目指すものである．

クラスタ環境では複数のアプリケーションが同時に動作していると考えられるため，電源管理システムは，複数の要求が同時に発生した場合でも破綻なく制御が行えることが必要である．たとえば，計算には使わなくなったノードがあったとしても，別のアプリケーションがデータスト

レージとしてそのノードを利用している場合は，そのノードと，データ転送に必要なネットワーク機器については電源を切らないで動かしておく，という処理が必要になる．

むろん，VM 実行環境などの基盤ソフトウェアにとっても，アプリケーションの要求と協調した，破綻のない電源管理は必要な機能となる．提案システムは，これらの基盤環境からも利用されることが想定される．

2.2　インタフェース

前述の要求をふまえ，アプリケーションや基盤ソフトウェアから容易に利用できるインタフェースを目指して設計を行った．

要求は「サービス」と「ノード名」により指定されるものと整理した．プロセスを起動したい，ローカルストレージのデータを参照したい，などの要求項目が「サービス」として想定される．この 2 情報を与えると，システムは計算ノード，外付けストレージ，ネットワークスイッチなどサービス提供に必要な機器をリストアップし，それぞれの機器に対応した電源管理システム（IPMI，SNMP など）を適切に利用して，自動的に機器の電源を ON にする．そのため，ユーザはネットワークトポロジなどの詳細な機器構成について知る必要がなく，また電源管理方式の違いについても考慮せずに統一的に扱うことが可能である．

インタフェースはコマンド実行によるものとした．現在の実装では，「ssh でユーザのプロセスが起動できる」というサービスのみを対象として実装している．内容を以下に示す．

- 計算ノードを指定した使用要求
  「*nodeid* の計算機」に「ssh したい」という要求を伝える．コマンドラインでは
  `% booking use` *nodeid*
  と実行する．システムは指定された計算ノードのほか，動作に必要なネットワーク機器を認識し，必要に応じて電源を ON にする．
- 計算ノードを指定した解放要求
  使用要求を出した計算ノードの利用を終えたら，ユーザは解放要求を出す．誰も使用しなくなった状態が一定時間（現在の運用では 20 分）続いた計算ノード，ネットワーク機器は，自動的に電源を落とされる．コマンドラインは
  `% booking release` *nodeid*
  である．
- 情報取得
  使用要求が出されている計算ノード，要求された時刻，使用者の情報が取得できる．また，機器の消費電力を提示するインタフェースも備えている．これは，計算で利用する消費電力の最大値にユーザが制約を加えたい場合などに利用することを想定している．

また，ユーザが計算ノードの解放を忘れることはしばしば起こると考えられる．これを避けるため，1 日 1 度，一定の時間になるとある時刻以前の使用要求を削除するという運用を行った．削除時間をまたいで計算機を利用したいユーザは，削除時間に再度使用要求を出し，計算ノードの要求を更新する必要がある．我々のクラスタは関係者数十人で利用しているが，これまでのところ，このような運用でそれほど大きな支障は発生していない．

2.3　実装

クラスタの各機器は，様々に異なる手段で電源管理を行う必要がある．計算サーバについては，サーバに内蔵されている IPMI による電源管理を利用するとともに，前述した PDU のコンセントごとの電源制御も補助的に利用する．PDU の制御は SNMP による．ネットワークスイッチは SNMP を用いた PDU の制御を利用する．また，データ用 RAID アレイも PDU を用いた制御を行えるように実装した．なお，電源管理においては，サーバの電源を落とすだけでなくサスペンドやハイバネーションを利用するという方法も考えられるが，今回の実験に用いた計算機ではサスペンドなどは利用できなかったため，電源の ON/OFF にのみ対応する実装としている．

システムはサーバ，ネットワークなどすべての管理機器について，接続関係に関する情報を保持する必要がある．指定されたサービスを実現するために必要な機器について，依存関係をたどることで制御が必要な機器の集合を抽出することが可能になる．現時点では，計算サーバとネットワークスイッチの単純な依存関係のみを扱えるように実装を行った．

3.　電力量削減効果の評価シミュレーション

計算機環境の電源に関しては，ピーク電力の削減など様々な要請があるが，本稿では総消費電力量を削減することを主に想定している．利用がない計算機を一定時間後に停止する，という提案手法が，どの程度電力量削減に貢献できるのかは調査する必要がある．サーバの起動停止を繰り返すことで，サーバが利用できない無駄な時間が生じ，かえって電力量が増加することも考えられる．そこで，シミュレーションを行って評価を試みた．

3.1　投入されるタスクのモデル

クラウド環境は様々なアプリケーションによって利用されるため，需要の推移についても多様な場合が考えられる．ユーザに対しサービスを提供している場合には，昼と夜などの時間変化にともなうユーザ数変化や，イベント発生時の突発的な需要などの増減要因が考えられる．研究開発に利用されている計算機では，異なる要因により異なる増減傾向を示すと考えられる．

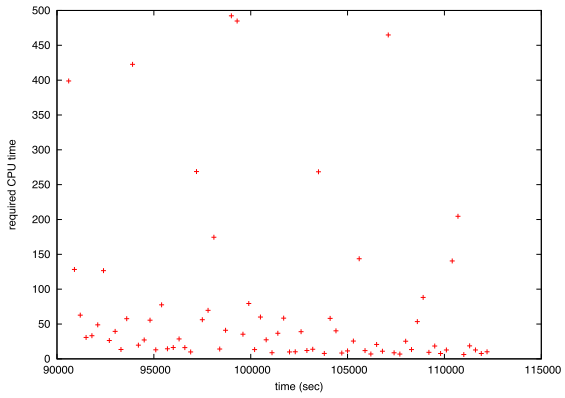

図 2　投入タスクの CPU 要求量
Fig. 2　Required CPU resources of task sequence.

本稿では，Google 社が所有する大規模なクラスタでのタスク実行トレース [7] を用い，計算機に投入される需要のモデルとして利用することとした．公表されているトレース[*1]のうち，TraceVersion1 を利用することとする．このデータは，Google の所有するクラスタ上で 7 時間の間に実行された 18 万個程度のタスクについて，各時刻時点での使用リソース量などのログをまとめたものである．外部へのサービスを行うレベルのタスクから，研究開発時のタスクまでが入り交じったものであり，実世界の大規模クラスタにおける需要の時間変化をよく反映していると考えられる．

ここでは，タスクが最初に投入された時刻と，タスクごとに集計した全使用 CPU リソース量に着目する．時刻ごとに，その時点で初めて投入されたタスクの使用 CPU 量を集計したものを**図 2** に示す．なお，CPU リソース量に関しては正規化が行われており，ログからは相対値のみしか読み取れない．横軸はログ上の時刻（秒）であり，90,000 秒から 112,500 秒までのデータが存在した．多くの時点においてそれほど需要は高くないが，時折突発的に高い需要が発生していることが見て取れる．実際にはログ開始直後の 90,300 秒時点で，他とは桁の違う，23,000 を超える CPU 量のタスクが 1 度に投入されているが，ここでは，平常時においてタスクがどのような頻度・分布で投入されるのか，に興味があるため，本実験においてはこのような特殊ケースを外し，ログの 90,600 秒時点以降のタスクのみを用いてシミュレーションを行うこととした．

本提案システムの制御方式は，電源の ON/OFF を頻繁に行うほど無駄な電力消費が発生するため，ある時点の需要が前の時点に対してあまり異ならないという性質，すなわち時間的局所性がある場合には，無駄な ON/OFF が減ってより大きな電力量削減効果が得られると考えられる．この観点から図 2 を見ると，投入されるタスク量は時間とともに大きく変動し，あまり傾向は見られない．しかし，多くの時間に投入されているのは比較的小さいタスクであり，多くの時点で需要は小さい，という点で時間的局所性が現れ，電力量削減効果が得られる可能性がある．

[*1] http://code.google.com/p/googleclusterdata/

3.2　シミュレーション

前述のタスクが，100 台の計算機からなるクラスタに投入される場合を想定し，タスク実行状況と消費電力のシミュレーションを行った．ここで，サービスレベルの要求として，どのタスクも 10 分以内に終了することが望まれていると仮定した．投入されるタスクの最大需要は 500 単位程度であるので，500 単位のタスクが計算機 100 台で実行されると 10 分かかるものと設定した．1 ノードの計算機が 1 分計算するとタスクは 0.5 単位実行されることになる．それぞれのタスクは投入時刻になると 1 度にシステムのキューに入る．計算機は停止（off），起動中（boot），アイドル（idle），計算中（busy），シャットダウン中（shutdown）のいずれかの状態にあり，システムは各時刻ごとにアイドル状態の計算機にタスクを割り当てる．計算機の起動停止について，off 状態から起動する場合，boot 状態に 3 分間とどまり，その後 idle 状態に遷移する．また，shutdown 状態は 1 分間とし，その後 off 状態に遷移する．これらの時間は，表 1 の計算環境での動作を参考に設定した．また，各状態での計算機の消費電力は表 2 の実測結果値とし，ネットワーク機器の消費電力はここでは考えないこととした．シミュレーションの時間刻みは 1 分単位としている．並列実行粒度に関しては，タスクは 1 ノードで 1 分の計算で終了する 0.5 単位のサブタスクから構成されていると想定する．

また，本稿で考慮しているアプリケーションは Elastic な動作を行うもの，すなわち，動作中に計算機構成を自ら変更しながら計算を行うものである．この前提に基づき，システムは idle 状態のノードをあらゆる時点で最大限の数だけ利用しようとすると仮定した．この場合，タスクは idle 状態のノードが 1 台でもあれば実行開始される．一般のバッチキューシステムのスケジューリングとは異なり，必要な並列実行ノード数が確保できるまで待つという動作は行わない．

システムの電源管理方式として，大きく 3 通りの場合を考える．

- 管理なし（not elastic）：計算機の起動停止を行わない．つねに 100 台が稼働状態（idle，busy）にある．
- 増減管理あり，タスク投入時制御（elastic，not dynamic）：計算機を増減させる．タスク投入が起きる時点で，そのタスク量を 10 分で処理できると考えられるだけの計算機数を求め，その数まで計算機を起動する．また，idle 状態が 20 分以上継続した場合，計算機を停止する．
- 増減管理あり，実行時制御（elastic，dynamic）：前述のタスク投入時の制御に加え，現時点で投入されて

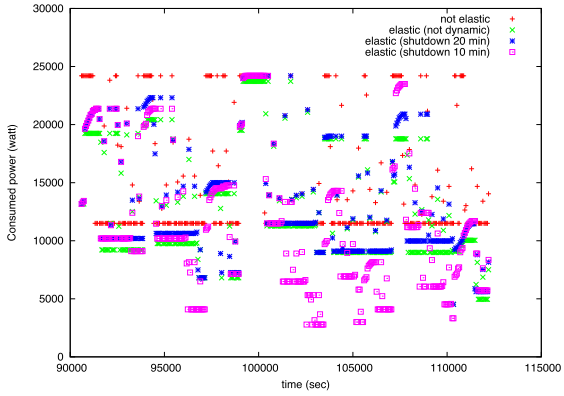

図 3　消費電力
Fig. 3　Power consumption.

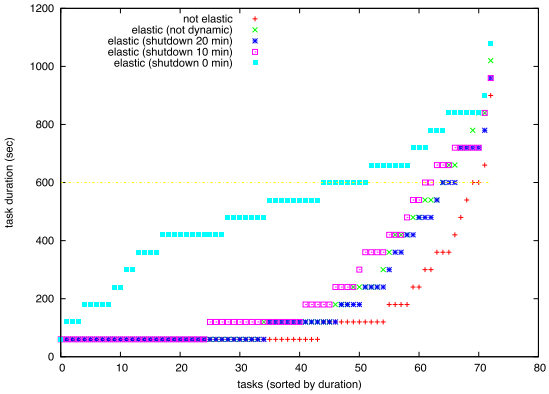

図 4　タスクごとの所要経過時間
Fig. 4　Duration time of each task.

いる残存タスク量を，現時点で利用可能（boot，idle，busy のいずれか）な計算機数で 5 分以内に処理できない場合には，計算機を 1 分につき 1 台ずつ起動する．先行するタスクがたまっている状況で，リソース不足になるのを避ける目的がある．この方式については，idle 状態で何分待つかという計算機停止制御の閾値を 20 分，10 分，0 分と変更しての評価も行った．

また，elastic な方式では，最初の時刻では 20 台の計算機のみが idle 状態，残りは off 状態にあるとした．

それぞれの制御方式で実行をシミュレートしたときの，クラスタ全体の消費電力の時刻推移を**図 3** に示す．横軸が時刻，縦軸がその時点での消費電力（W）を示す．見やすさのために「elastic（shutdown 0 min）」の実験結果については省略してある．not elastic な場合には，ほぼすべての計算機が idle な状態ですごしている時間（消費電力 12,000 W 近辺）が相当量存在することが見て取れる．elastic な制御を行うと，必要に応じて計算機を停止するため，より電力消費を抑えられる期間が生まれている．タスク投入時の制御のみでもノード増減は実現されているが，実行時情報を用いた制御を利用すると，時折状況に合わせてノード数を増加させてサービスレベルを保とうとする様子が見て取れる．

タスクごとに，投入時刻から計算終了時刻までの所要経過時間を求めたものを**図 4** に示す．横軸はタスクを表し，所要時間が小さいものから順にソートされている．not elastic の場合，目標はほぼすべての場合で満たされている．一部の 600 秒超えのタスクについては，短時間に大量のタスクが投入されているため，今回想定した計算機のキャパシティでは要求が満たせなかったことが分かる．elastic な制御を行うと目標を満たせないタスクが増加するが，not dynamic な場合に比較して，dynamic 制御を行うとタスク所要時間が若干高速化されているのが見て取れる．目標を満たせなかったタスクの割合を計算すると，11%から 8%に減少しており，よりサービスレベルを守る方向で制御が行えていることが確認できる．計算機停止までの閾値時間を短くしていくと，ノードの停止再起動が頻発するため，利用できる CPU 時間が減少し所要時間は悪化していくことも見て取れる．

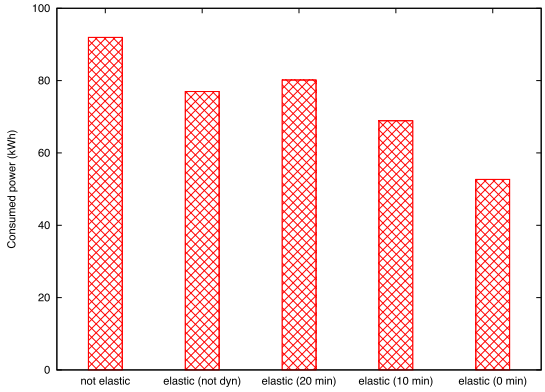

図 5　積算消費電力量
Fig. 5　Power consumption (accumulated).

それぞれの制御方式において，期間全体で消費された積算電力量を求めたものを**図 5** に示す．制御しない場合と比較して，elastic に電源制御を行うと総消費電力量が抑えられることが確認できる．停止までの閾値を 20 分とした場合，タスク投入時のみの制御で 17%，実行時情報による制御を加えたときで 13%程度の電力量削減が実現された．実行時情報制御方式の場合は，サービスレベルの低下も抑えられており，有効性が示されたといえる．また，ノード停止までの閾値時間を短くしていくと，消費電力量もより削減される結果となったが，図 4 のようにサービスレベルとのトレードオフが存在するため，アプリケーション特性に合わせて適切な閾値を設定する必要があると考えられる．

なお，本評価では定常状態のシステムの性能比較を行うことが主目的であるが，シミュレーションを行った時間範囲を考慮すると，積算電力量の比較に際して開始時と終了時の過渡状態が占める割合は無視できる程度であると考えている．

4. 実環境での実証評価

提案手法を実装したクラスタ上で，実アプリケーションを用い，需要に応じた動的なノード数制御を行うことで，消費電力量を削減することを試みた．

4.1 GXP における電力制御システムとの連携

提案する電源管理システムをプログラム処理系から利用して電力制御を行うことが容易であることを確認するため，プログラム処理系の 1 つとして，分散タスク処理系 GXP [8] を用いて連携実験を行った．GXP はオンデマンドに確保した計算機を即座に利用して分散計算を行うことに注力した処理系であり，クラウド環境を強く意識しているといえる．

GXP は，分散計算環境において複数の計算機を自由に選択しつつシェルのコマンドを発行することができる，分散シェルの機能を基本とする実行環境であり，ユーザプログラムや make コマンドと組み合わせることで，タスクキューイング，マスタワーカ型，ワークフロー，などの形態の分散処理を行うことが可能な処理系である[*2]．コマンドを実行する計算機をユーザの指示によって追加・削除する機能を持つが，需要に応じたノード数制御など，Elastic Computing を支援するための高レベルな機能は持たない．そこで，GXP を外部から制御して需要への適応と電力削減を行えるようなスケジューラを作成し，実験に用いることにする．

GXP には，ソケットやパイプなどのストリームを通してタスク投入を行うインタフェースが存在する．ここでは，そのインタフェースに

- 計算機構成，タスク処理状況などの情報取得
- 動的な計算機の参加・脱退

を可能とするような変更を加え，GXP の外部にスケジューラプロセスを別途作成し，接続した．

追加されたスケジューラは，**図 6** の擬似コードが示すようなアルゴリズムに従い，電源管理システムと連携して Elastic な計算を実現できるような動的並列度制御を行う．擬似コードで，*start_h* は起動中と考えられる計算機集合，*running_h* は GXP によって利用されている計算機集合，N は現在スケジューラが目標としている並列度である．以下に動作の詳細を示す．

- 利用できる計算ノードのリストを事前に取得する．
- 初期並列度の「ヒント」があればそれを取得する．(図 6 (a))
- 実行に必要な並列度の数だけリストから計算ノードを選択し，電源管理システムを呼び出してその計算ノードを利用する希望を伝える．(b)

[*2] GXP の利用法に関しては，http://www.logos.t.u-tokyo.ac.jp/gxp/のチュートリアル資料などが詳しい．

```
start_h ← ∅, running_h ← ∅
N ← 初期並列度ヒント                                   (a)
while GXP が終了していない:
  if |running_h ∪ start_h| < N:                        (b)
    h ← 足りない台数分の利用可能ノード
    h のノードを利用開始 (booking use)
    start_h ← h
  forall m in start_h:                                 (c)
    if m が起動した:
      m を GXP に通知
        (GXP は即座に m を利用開始)
      m を start_h から running_h へ
  if GXP の残りタスク数 > 閾値:                        (d)
    N ← N + 1

forall m in running_h ∪ start_h:                       (e)
  m の利用終了 (booking release)
```

図 6　Elastic スケジューラのアルゴリズム
Fig. 6　The algorithm of Elastic Scheduler.

- 定期的に計算ノードの状態をチェックし，ノードが利用可能になっていたならば GXP の分散計算環境に追加する．(c)
  GXP にとってはタスク実行可能な計算機が動的に増えたことになる．このとき，GXP は並列度が増えた状態で即座にスケジューリングを行い，並列計算全体は止まることなく実行され続ける．
- 定期的に GXP の持つタスクキューの長さをチェックし，閾値以上のタスクが未実行状態のままであれば，新たに計算ノードを 1 台追加する．(d)
- タスクキューにタスクがなくなり，GXP がすべてのタスク処理を終了したら，電源管理システムにすべての計算機を解放することを伝え，終了する．(e)

この制御は，3.2 節のシミュレーションにおいて，実行時制御を加えた電源管理方式に相当する．また，初期並列度のヒントはタスク投入時の制御に相当する．

今回の変更において，GXP を利用するアプリケーションプログラムの方は基本的には特に変更を要しない．投入されるタスクの数に応じて GXP が自動的に実行計算機数を調整する．また，python で記述された GXP に対してインタフェース追加のために加えた変更は 60 行程度，作成したスケジューラは python で 260 行程度の規模となった．いずれの実装もそれほど大きな規模ではなく，提案システムの利用は容易であったと考えている．これは，もともと GXP が動的な構成変更を想定し，計算機の参加脱退などの必要な機能を備えていたことに起因する部分が大きい．Elastic な計算を想定したプログラム処理系は，これらの機能を備えているはずであり，同程度の労力で同様の制御方式を実装することが可能であると考えられる．

本実験では，スケジューラプロセスを GXP の外部に作成したため，スケジューリングに関する処理は GXP の実装とは切り離されており，他のプログラム処理系で同様の処理を行う際に共通的に利用できる部分は多いと考えられる．他の処理系においても，プログラム処理系外部から需要と処理状況を観察し，処理系にリソース増減などの指示を出す，というスケジューラを作成することで，たとえばウェブアプリケーションではロードバランサによるウェブサーバ数の制御，Key-Value ストアではリクエストレート監視によるキャッシュ数の制御など，本実験と同様の電力制御が容易に可能になると期待される．

4.2　探索アプリケーション

実アプリケーションの例として，コンピュータ将棋プレイヤ「激指」[9] を用いた．将棋の探索は局面によって訪問すべき探索木の大きさが異なり，必要となる計算量が大きく変化する．我々は，計算量予測に基づき動的に計算に用いるリソース量を変化させることで，サービスレベルを守りつつ，必要最低限のリソースを利用して計算を行う手法を提案している [10]．本稿ではこのアプリケーションを，需要に応じた構成変更が行われる例題として利用し，提案手法を用いることで消費電力量を削減できるかの評価に用いることにした．

激指は GXP をワークフローエンジンとして利用し，分割したゲーム木の探索を独立したタスクとして処理系に投入，結果を集約して木全体の探索結果を得る，という分散計算を行う．本実験では，ゲーム木をおよそ 300〜10,000 個程度に分割するような設定を用いた．

4.3　棋譜解析時の消費電力量評価

このソフトウェアを用い，対局中継中の形勢解析サービスを想定したアプリケーションを考えることにする．将棋の対局はしばしば中継されており，一手指されるたびに，形勢やその後の展開などを予想してリアルタイムに検討する，という楽しみ方がある．人間同士の対局では，一手ごとの経過時間は様々であり，サービス需要が不定期に発生することになる．前述のように，局面ごとに検討に必要な CPU リソース量も大きく異なる．一方，コンピュータによる検討結果は，どのような場合であっても一定時間内に返されることが望ましい．このような観点から，形勢解析アプリケーションは，Elastic な計算環境でサービスレベル制約を考慮しつつ実行されるべきものであるといえる．

実験は，2010 年に行われた強豪将棋ソフトウェア対強豪女流プロの対局イベント，「清水市代女流王将 vs. あから 2010」[*3]での対局結果を用い，対局中に現れたそれぞれの局面（全 86 局面）で一定の深さまで探索することとした．サーバは 32 台までを利用することとし，制御方式として，

- const：つねに 32 台を利用する場合．
- static：各局面の探索に必要な時間をあらかじめ測定しておき，最も時間がかかる局面で 45 台を利用希望するように線形に必要台数を予測し，初期並列度ヒントを与えた場合（なお，実際に計算に利用できるのは最大でも 32 台である．ヒューリスティックに基づき，少し多めの台数を利用するように制御することを意図している）．
- dynamic：探索総時間予測 [10] を行い，それをもとに 32 台までの範囲でヒントを与えた場合．

の 3 通りについて計測を行った．static はすべてのタスクの必要リソース量があらかじめ分かっているアプリケーションを仮想的に考えた場合に実現できる手法であり，dynamic は，不正確ではあるがリソース量がある程度予測できる実アプリケーションで利用できる制御手法である．なお，一手の探索が終わるごとに GXP は終了し，電源管理システムに全計算機の利用終了が通知される．

static，dynamic については実験開始時には 3 ノードの計算ノードのみ電源が入っているものとし，const は実験開始時から終了時までつねに 32 ノードが稼働状態であるとした．const に関しては，タスクがいつ投入されるのかを事前に知っていれば，タスクが投入されるタイミングに合わせて 32 ノードを ON にし，1 つのタスクを全力で処理したらすぐさま電源を切る，という手法も考えられるが，本稿ではタスクの投入時期が分からない状況を想定しているため，タスク投入の直前に電源を ON にすることは考慮しにくい．また，タスク投入後に電源を ON にした場合には，計算ノードの起動に約 3 分かかる環境であるため（1.2 節），2 分程度の計算時間に対してオーバヘッドが大きすぎると考えられ，現実的な解ではない．なお，static，dynamic の方法に関しても，制御の結果完全にすべての計算機が停止してしまう，ないしは到着するワークロードに対してきわめて少ない台数しか稼働していないような時間が発生した場合には，同様の問題が起きると考えられる．これを防ぐためには，最低限の台数だけの計算機は稼働させたままにする，という扱いをする必要があるが，何台の計算機を稼働状態で残しておくのかはワークロードの性質と SLA の関係によって決めるべき問題であり，単純に決定することは難しい．本実験では，少なくとも 1 台の計算機はつねに稼働状態にするという実装を用いた．また，今回用いたワークロードと制御ポリシでは，最も稼働台数が少なかったのは 3 台で計算開始した実行開始時点であり，つねに 3 台以上の台数が稼働した状態であった．

図 7 は，それぞれの方式での各局面での探索時間を示している．static はほぼすべての局面で 60 秒から 120 秒の範囲で探索が終了しているが，const は探索時間が短い局面と長い局面の差が大きい．dynamic は必要なリソース量

[*3] http://www.ipsj.or.jp/50anv/shogi/index2.html

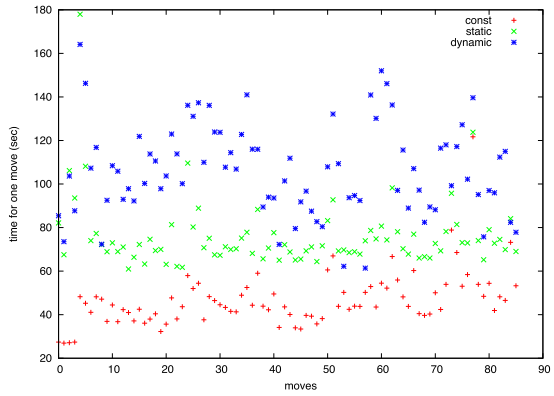

図 7　手ごとの消費時間
Fig. 7　Execution time of each move.

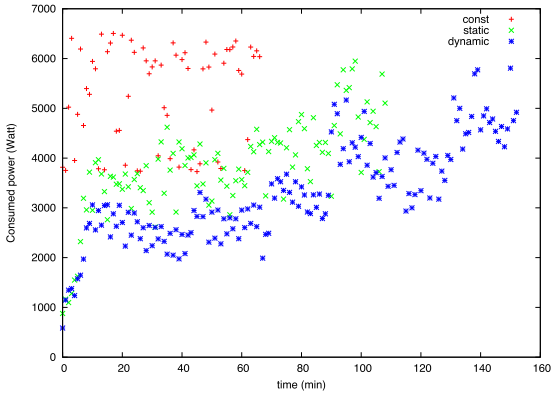

図 8　消費電力
Fig. 8　Power consumption.

の予測が不正確なため，static よりは探索時間が延びている．また，const を見ると各局面で必要な計算量の変化がおおむね把握できる．一手前の局面に対して比較的近い計算量が必要になる傾向があるように見受けられ，効率良く電力量を削減できる可能性があると考えられる．

それぞれの手法について，消費電力の時間変化を測定した結果を**図 8** に示す．横軸は実験開始時からの経過時間（分）であり，1 分ごとに消費電力を測定した結果を縦軸にプロットしてある．計算ノードについては実測値を用いたが，ネットワークスイッチに関しては，トポロジの違いによる変化を避けるため，仮想的に 16 ノードずつ 2 つのスイッチに接続された環境を想定し，消費電力を計算した．図 8 では，計算ノードとネットワークスイッチの双方の消費電力の和を示している．const はつねに 32 ノードの計算ノードを利用しているため，使用電力の変化は CPU の DVFS などによる制御に起因している．比較的多くの時間において，4 kW 程度の消費電力で稼働しているが，このときは計算需要と比較して投入リソース量が多すぎ，アイドルな計算機が多数発生していると考えられる．static，dynamic は総探索時間が const より延びているが，探索途中の消費電力は低く抑えられている．

図 8 の消費電力時間推移をもとに，計算全体で使用した積算電力量を求めたものを**図 9** に示す．busy の系列が，各々の制御手法で実際に計算を行っている時間内における消費電力量を示している．static や dynamic は，const と比較して消費電力量が増えている．本アプリケーションでは，探索全体で必要となる CPU 時間は台数の違いによらずほぼ一定であるため，この電力量の差は，主にノードの起動や停止にともなって発生する無駄な稼働時間に起因するものと考えられる．

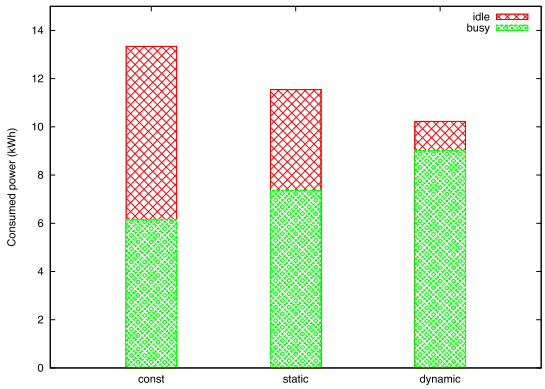

図 9　積算消費電力量
Fig. 9　Power consumption (accumulated).

ここで，リアルタイムに進行する対局を随時解析するようなアプリケーションを考えると，棋譜は 1 度に与えられるのではなく，手が指されるたびに解析需要が発生することになる．ここでは，単純化のために一定時間ごとに手が指されたという仮定をおく．最も時間がかかる局面を 32 台の計算リソースで解析するのに約 2 分を要するので，2 分ごとに一手ずつ手が指され，そのたびに解析を行ったと想定すると，一手ごとに解析を終えた後は利用した計算ノードがアイドル状態で次の手を待つことになる．この待ち時間に消費した電力量を計算したものを図 9 の idle の系列に示す．const の場合は，32 台の計算機と 2 台の 10G スイッチが表 2 の idle 状態にある場合の消費電力量を計算した．static，dynamic については，一手各々の解析終了時に使用していた電力が idle 時間においてもそのまま継続して消費されたと仮定して計算を行った．これは，static，dynamic の手法における最大消費電力量に近い見積りであり，実際には数割程度少ない消費電力量になると期待される．また，手は一定間隔で到着するため，一手の解析に 2 分以上かかる場合には次の手のアイドル時間が削られるものとして計算している．アイドル時間の消費電力も含めて比較すると，Elastic な制御を行った方が全体の電力量が削減されていることが分かる．本想定では，制御を行わない場合に対し，完全な需要予測が行える（static）場合はおよそ 14%，不完全な需要予測（dynamic）の場合には 23%程度の電力量削減が実現できた．ただし，アイドル時間の消費電力見積りが大きめにとってあるため，実際には static と dynamic のどちらが省電力なのかは逆転している可能

性もある．また，dynamic な制御では 2 分以内に解析が終わらない局面が多数発生しており，いわば想定 SLA を守ることができていない状態にあることも考慮する必要がある．いずれにせよ，ここで想定している解析サービスアプリケーションのように，変化する需要が時間経過とともに与えられるような場合には，本提案のような電力制御と連携した計算手法が有効であると考えられる．

5. 関連研究

CPU における Dynamic Voltage and Frequency Scaling（DVFS）[3] などの機能，ディスクのスピンダウン機能など，個々の計算リソースにおける電力削減の試みは進んでいる．これらの機能を利用する試みも多数あり，HPC [11]，ウェブサーバ [12]，データベース [13] など，様々なアプリケーションの省電力化に関する研究が行われている．これらの機能を組み合わせた際の効果に関する研究 [14] も行われている．クラスタ全体での電力制約がある状態で CPU インテンシブアプリのパフォーマンスを最大化する研究 [15] も存在する．

クラウド環境のスケジューリングに関する研究は，複数ワークロードの混在を考慮したもの [16]，バースト的に生じるワークロードを考慮したもの [17] など多数存在する．これらは制御の目的をクラスタ利用効率の向上においているが，電力を考慮したスケジューリングについても研究が増えている．Zhu ら [5] は，科学計算に多く用いられるワークフローを対象とし，CPU やディスクなど異なるリソースがボトルネックになるタスクを集約して実行することで，消費電力量を抑えるスケジューリングを提案している．また，von Laszewski ら [6] は，DVFS が利用できるクラスタで VM の配置を最適化する手法を提案している．Zhang ら [18] は，消費電力量を考慮した計算機キャパシティ制御を行っており，同様の問題意識を持つ．これらの研究では，ワークロードはコントロールできないものとして扱われており，与えられたワークロードをどのように処理するかという問題に注力している．本稿は，プログラム処理系が自ら Elastic Computing のために使用リソースを変える場合にも，同様に消費電力量を考慮すべきである，という観点から，必要な電力管理システムの姿について議論を試みたものであり，前述のような関連研究の制御手法はプログラム処理系自身が行う電力制御の方法設計に大いに活用されるべき知見であると考えられる．

6. おわりに

Elastic な処理を要求する計算需要が存在しており，クラウドコンピューティング環境を活用してそのようなワークロードに対応しようという動きが広まっているが，リソースコストの最小化だけでなく，消費電力量の削減についても考慮が求められ始めている．本稿では，一定時間必要でないとされた機器を自動的に停止することで消費電力量を削減する手法を提案した．機器停止においては，個々の計算環境における機器構成を理解し，ネットワークなどの設備も含めて一貫した管理が必要となる．我々は，実際のクラスタ上でこのようなシステムを設計実装し，長期にわたって運用を行っている．この経験から，ユーザが提案インタフェースを利用することは容易であり，プログラム処理系などからの連携が期待できると考えている．

提案手法の有効性を評価するため，大規模実クラスタ上で取得されたタスクトレースを用いたシミュレーションを行ったところ，サービスレベルをある程度保った状態で13%程度の電力量削減が可能であるとの結論を得た．また，電力を考慮した制御を加えた探索アプリケーションを構築し，実際のクラスタ上での電力測定を行ったところ，需要予測が正確な場合でおよそ 14%，不完全な場合で 23%程度の電力量削減が実現できた．

本稿の電力制御システムはまだ必要最低限の機能しか持っておらず，より複雑なリソース依存関係の取扱いなど，様々な改良が必要である．また，バッチキューシステム，分散ストレージシステムなど，より多くの計算サービスとの連携実現を模索していくことを通して，電力制御システムとサービス提供システムの双方が満たすべき機能，などを明らかにしていくことが必要である．今後の課題としたい．

謝辞　本研究の一部は科学研究費助成事業（24700023）の助成によって行われた．

参考文献

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M.: A view of cloud computing, *Comm. ACM*, Vol.53, No.4, pp.50–58 (online), DOI: 10.1145/1721654.1721672 (2010).

[2] Barroso, L.A. and Hölzle, U.: The Case for Energy-Proportional Computing, *Computer*, Vol.40, No.12, pp.33–37 (online), DOI: 10.1109/MC.2007.443 (2007).

[3] Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z. and Zhu, X.: No “power” struggles: Coordinated multi-level power management for the data center, *ASPLOS XIII*, pp.48–59 (online), DOI: 10.1145/1346281.1346289 (2008).

[4] Abts, D., Marty, M.R., Wells, P.M., Klausler, P. and Liu, H.: Energy proportional datacenter networks, *Proc. 37th Annual International Symposium on Computer Architecture*, *ISCA ’10*, pp.338–347 (online), DOI: 10.1145/1815961.1816004 (2010).

[5] Zhu, Q., Zhu, J. and Agrawal, G.: Power-Aware Consolidation of Scientific Workflows in Virtualized Environments, *SC 2010*, pp.1–12 (online), DOI: 10.1109/SC.2010.43 (2010).

[6] von Laszewski, G., Wang, L., Younge, A. and He, X.: Power-aware scheduling of virtual machines in DVFS-enabled clusters, *IEEE International Conference on Cluster Computing and Workshops* (*CLUSTER ’09*), pp.1–10 (online), DOI: 10.1109/CLUSTR.2009.5289182

(2009).
[7] Hellerstein, J.L.: Google cluster data, Google research blog (2010), available from ⟨http://googleresearch.blogspot.com/2010/01/google-cluster-data.html⟩.
[8] Taura, K.: GXP: An Interactive Shell for the Grid Environment, *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, pp.59–67 (2004).
[9] 激指開発チーム：将棋プログラム「激指」，入手先 ⟨http://www.logos.ic.i.u-tokyo.ac.jp/~gekisashi/⟩.
[10] 横山大作，喜連川優：クラウド環境での分散ゲーム木探索実現への取り組み，第 3 回データ工学と情報マネジメントに関するフォーラム（DEIM2011），pp.C3–5 (2011).
[11] Hsu, C.-H. and Feng, W.-C.: A Power-Aware Run-Time System for High-Performance Computing, *Proc. 2005 ACM/IEEE Conference on Supercomputing, SC '05*, p.1, IEEE Computer Society (online), DOI: 10.1109/SC.2005.3 (2005).
[12] Wang, Y., Wang, X., Chen, M. and Zhu, X.: PARTIC: Power-Aware Response Time Control for Virtualized Web Servers, *IEEE Trans. Parallel and Distributed Systems*, Vol.22, No.2, pp.323–336 (online), DOI: 10.1109/TPDS.2010.79 (2011).
[13] Nishikawa, N., Nakano, M. and Kitsuregawa, M.: Energy Efficient Storage Management Cooperated with Large Data Intensive Applications, *IEEE ICDE 2012* (2012).
[14] Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P. and Zhu, X.: Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble, *Proc. Conf. on Power Aware Computing and Systems* (*HotPower*), *HotPower'08*, p.2, USENIX Association (online) (2008), available from ⟨http://dl.acm.org/citation.cfm?id=1855610.1855612⟩.
[15] Wang, X. and Chen, M.: Cluster-level feedback power control for performance optimization, *IEEE 14th International Symposium on High Performance Computer Architecture, 2008, HPCA 2008*, pp.101–110 (online), DOI: 10.1109/HPCA.2008.4658631 (2008).
[16] Tumanov, A., Cipar, J., Ganger, G.R. and Kozuch, M.A.: alsched: Algebraic scheduling of mixed workloads in heterogeneous clouds, *ACM Symposium on Cloud Computing* (*SoCC*), pp.25:1–25:7, ACM (online), DOI: 10.1145/2391229.2391254 (2012).
[17] Ali-Eldin, A., Kihl, M., Tordsson, J. and Elmroth, E.: Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control, *Proc. 3rd Workshop on Scientific Cloud Computing Date*, *ScienceCloud '12*, pp.31–40, ACM (online), DOI: 10.1145/2287036.2287044 (2012).
[18] Zhang, Q., Zhani, M.F., Zhang, S., Zhu, Q., Boutaba, R. and Hellerstein, J.L.: Dynamic energy-aware capacity provisioning for cloud computing environments, *Proc. 9th International Conference on Autonomic Computing*, *ICAC '12*, pp.145–154, ACM (online), DOI: 10.1145/2371536.2371562 (2012).

横山 大作　（正会員）

2006 年東京大学より博士号取得．博士（科学）．2002 年より同大学新領域創成科学研究科助手等を経て，2009 年より同大学生産技術研究所助教，現在に至る．並列プログラミング，分散計算環境，ゲームプログラミングに関する研究に従事．

喜連川 優　（フェロー）

東京大学大学院工学系研究科情報工学博士課程修了（1983 年）．工学博士．現在，国立情報学研究所長，東京大学生産技術研究所教授，東京大学地球観測データ統融合連携研究機構長．文部科学省「情報爆発」特定研究領域代表（2005～2010 年），経済産業省「情報大航海」戦略会議委員長（2007～2009 年）．データベース工学が専門．ACM SIGMOD Edgar F. Codd Innovation Award 受賞，ACM/IEEE フェロー，本会/電子情報通信学会フェロー，本会会長，内閣府最先端研究開発支援プログラムを推進．