電力を考慮したプログラミングのための システム構築に関する検討

横山大作[†1] 田浦健次朗[†2] 喜連川優[†1]

クラウドコンピューティングと呼ばれるオンデマンド型計算環境が普及するに伴い、需要に応じて使用リソースを変動させて効率よく計算を行えるプログラミング手法が求められてきている。様々な制約から、近年では計算所要時間を短くするのみならず、使用する電力量を削減することが重要な要求になりつつある。このためには、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。しかし、計算機器の電源管理は複数のインタフェース規格が混在しており、一般ユーザにとって取り扱いが煩雑である。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。本発表では、このシステムの設計と、ワークフロー処理系と組み合わせて利用した場合の効果について紹介する。

Towards Energy Efficient Programming – An Implementation of a Power Control Interface on a Cluster

Daisaku Yokoyama,[†1] Kenjiro Taura[†2] and Masaru Kitsuregawa[†1]

As on-demand computing environments have become popular, elastic programming frameworks that can adapt resources to demands have become to be required. Reducing electric energy consumption is highly required in these days, as well as speeding up the computation. We should minimize powered elements that are sufficient to execute user's job. Clusters are often built by several components such as computing nodes, interconnects, storages, that are controlled by the mixture of several de facto power APIs. These components should be treated in a coordinated way to achieve required job. It is not so easy to use. We propose a power API on a cluster that can control these components with simple and unified way. Our system can reduce power consumption through shutting down elements that are not required by the user's job. In this presentation, we describe the design and implementation of this system, and introduce a use-case working with a work-flow scheduler.

1. はじめに

近年、クラウドコンピューティングと呼ばれる計算環境が着目され、Amazon EC2 をはじめとする実用的クラウド環境の充実に伴い、広く使われ始めている。クラウドコンピューティングの重要な利点として、リソースのオンデマンドな利用が可能なことが挙げられる[1)]。状況によって大きく変化するリソース需要に対し、必要なときに必要なだけのリソースのみを利用して計算を行うことができ、それによって計算コストを削減できることがクラウドを利用する目的の 1 つである。

このような環境では、リソースを使わないときに固定的に必要になるコストをどれだけ削減できるか、ということが重要な考慮点になる。必要なときには大量サーバ、高性能ネットワークを利用した大規模計算を実現できるようにしつつ、必要がないときにはその環境の維持コストを最小化する、という設計が必要になる。

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押さえることが望ましいという考え方である。

†1 東京大学生産技術研究所
Institute of Industrial Science, The University of Tokyo
†2 東京大学情報理工学研究科
Graduate School of Information Science and Technology, The University of Tokyo

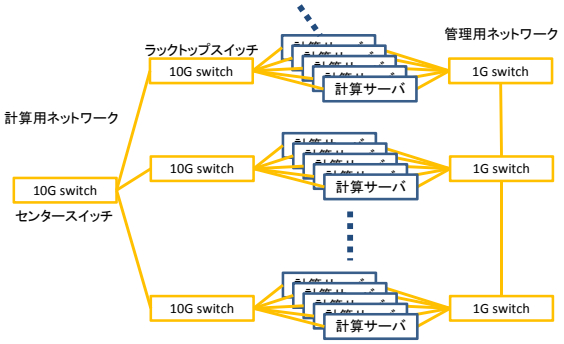

図 1 実験環境
Fig. 1 Experimental environment

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	24GB
Disk	500GB 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
10G switch	230
1G switch	48

計算環境の構成要素を考えたとき、CPU については、Dynamic Voltage and Frequency Scaling (DVFS)[3]、電源供給のブロック化などの技術により、計算を行っていないときの消費電力が大幅に削減されており、Energy Proportional な計算を実現しつつある。ディスクについてはアクセスが起きないときにスピンダウンして消費電力を押さえることができ、キャッシュを利用した省電力化などの研究も行われている[4]。また、SSD などの低消費電力なデバイスが盛んに研究されており、データ書き換え時以外に必要な電力を削減することが可能である。

このように、計算環境を構成するサーバ内部の個々の部品については、Energy Proportional な計算を実現しつつある。しかし、分散計算を行うクラスタのような分散計算環境全体を考えると、Energy Proportional という目標を達成することは容易ではない。

例として、我々が利用している計算機クラスタにおいて並列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**図 1** のように構成されており、

- 計算サーバ数: 127
- ラックトップスイッチ数: 7
- センタースイッチ数: 1
- 管理用スイッチ数: 4

だけの機器がある。また、それぞれの機器の詳細は**表 1** に示すとおりである。

本クラスタは Power Distribution Unit(PDU) として、Raritan DPXS12-30L-J (及びそのシリーズ) を利用しており、この PDU を用いて個々の機器毎に消費電力を測定することが可能である。これを用い、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 分間計測を行った時の平均値を示している。サーバについてはほぼ CPU のみを利用するアプリケーションを実行している時 (busy) と、OS 以外の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。なお、busy 時のアプリケーションは 3.2 の評価実験で利用したコンピュータ将棋プレイヤである。

ここで、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と、仮に全く計算を行っていないときとを考えると、それぞれの消費電力は**図 2** のように求められる。all busy が計算サーバを最大限利用し、全て busy であるとき、all idle が全ての計算サーバが idle であるときの想定消費電力である。計算サーバがアイドル状態でもクラスタの消費電力は半分程度にしかならず、計算サーバを停止するなどの方法をとることで大幅に電力削減が可能であることがわかる。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がわかる。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 状態にある場合を考えると、クラスタの消費電力は 4 nodes busy の系列に示される値になる。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わかる。これは、データセンター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[5] で述べられたものと同様の結果である。

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、消費電力の削減を図るような電源管理システムをクラスタ上に作成した。また、このインタフェースをユーザ

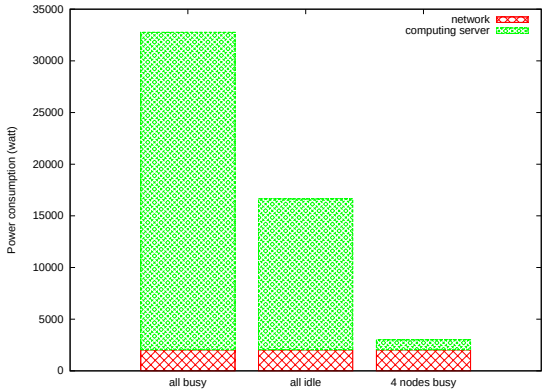

図 2 消費電力想定
Fig. 2 Estimated power consumption

に公開し、運用を試みた。

さらに、クラウド環境で求められる Elastic な計算を行うアプリケーションを題材に、プログラム処理系と電源管理インタフェースとの連携を行い、消費電力の削減を試みた。

本稿ではこの電源管理システム、並びに処理系との連携について紹介する。

2. 電源管理システム

計算環境の電源管理システムは、単に計算サーバの電源を ON/OFF できるような機能を提供するだけでは不十分である。ユーザに提供されるサービスは、計算サーバとネットワーク機器など複数の機器の連動によって実現されており、必要のないサービスを停止する際には複数の機器の電源管理を連動して行い、計算環境全体での電力最適化を図らなければならない。逆に、ネットワークを使うサービスが別に動いているにもかかわらず、ネットワークスイッチの電源を落としてしまうようなことも避けなければならない。

我々は、実験で使用しているクラスタ (図 1) 上に、このような一貫性を保った電源管理を行うシステムを構築した。

2.1 インタフェース

大きな方針として、ユーザは利用したいサービスを指定し、システムがそのサービスに必要な機器の電源を自動的に ON にする、というインタフェースを提供することとした。現在の実装では、「ssh でユーザのプロセスが起動できること」というサービスのみを対象として実装している。

インタフェースはコマンド実行によるものとした。内容を以下に示す。

- 計算ノードを指定した使用要求
  `% booking nodeid` のように、計算ノード名を指定して使用要求を出す。システムは指定された計算ノードの他、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。

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きると考えられる。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複数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。

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。ユーザが、計算で利用する消費電力の最大値に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。

2.2 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 を用いた制御が可能である。

システムはサーバ、ネットワーク等の接続情報をデータベースに持っており、サービス実現のための機器の依存関係をたどり、制御が必要な機器を抽出することができる。現時点での実装では、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。

2.3 議　　論

サービスとして現在は「計算ノードを指定して」「ssh での認証を受けてプロセスを立ち上げる」というものを考えている。しかし、サービスには他に様々なものが考えられる。

- 計算ノードを指定しない場合
  どのノードでも良いから 10 ノード、というような要求も多くあると考えられる。これは、バッチキューシステム的な利用でよく見られる。対応する必要があると思われるが、逆にこのような要求

しか許さない場合は、自由度が制約されすぎると考える。ローカルのリソースを利用したい (例えば、ローカルディスクに残っているはずのキャッシュを利用したい) ような時に、利用するリソースの明示的な指定が必要になると考えられる。

- サービスの中身と深く関わる指定
  例えば、分散ファイルシステムで「このファイルが置いてあるノードのみ使いたい」というような指定をすることが考えられる。この場合、分散ファイルシステムというサービスと密に連携し、ファイルシステム側から物理的なリソースを指定してもらう必要がある。
  このようなサービスは多数あると考えられるので、電源管理システムはクラスタ固有の機器構成に起因する基本的な依存関係のみを扱うこととし、各種サービスとうまく連携することで多様な要求を満たせるように構成することが必要であると考えられる。

電源管理システムの設計においては、これらの点に留意する必要がある。

3. Elastic computing 処理系との連携

クラウド環境を利用し、その利点を活用するためには、計算需要、あるいはユーザの要求に応じて動的に計算機構成を変更し、必要に応じた規模での計算を行う、Elastic computing を可能にするプログラミング環境が求められる。ここでは、そのような動的構成変更に対応している分散タスク処理系 GXP[6] を利用し、本提案の電力制御 API との連動を行った。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 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

- 現在の計算機構成、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修正を加え、GXP の外部にスケジューラプロセスを別途作成し、機器の電力制御と GXP の処理を連動させることを試みた。

3.1 Elastic スケジューラ

GXP に追加されたスケジューラは、以下に示す動作により Elastic な計算実行を行う。

- 利用できる計算ノードのリストは事前に設定される。
- 起動時に初期並列度の「ヒント」があればそれを取得する。
  アプリケーションはこれから行う分散計算がどの程度のリソースを必要とするか、ある程度の見積もりが行えることも多いため、アプリケーションの知識を用いて初期の並列度を設定することを可能にした。
- 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。
- 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。
- 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。初期の並列度の見積もりに誤差があり、十分な並列度があるにも関わらず利用しているリソース量が少ない場合においても、利用可能な空き計算ノードが存在するならばそのリソースを動的に利用することで、問題の大きさに追従した規模での分散計算を実現するためである。

現在は計算需要の指標として、GXP が保持しているタスクの数のみを利用しているが、平均タスク実行時間などを利用することでより正確に計算需要を把握することが可能になると考えられる。また、スケジューリングについても、あらかじめ指定した計算ノードが利用可能ならば全てを利用しようとするが、計算全体の電力量制約など、様々な制約条件を指定できるようにすることも必要であると思われる。

3.2 探索アプリケーション

Elastic な制御を要求する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」[7] を分散計算の適用ができるように修正し、本処理系を用いての制御を行った。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という特徴がある。例えば、同程度に有望と思われる指し手の候補が多くなる中盤から終盤の入り口あたりでは探索に必要な時間が多くなり、詰みに近くなった最終盤では探索時間が短くなる、という傾向がある。そのため、探索時間が短くてすむ序盤に合わせて投入する計算リソースの量を決めていると、中終盤においてリソースが不足し、人間の求めるサービスレベル (例えば、2 分以内に指してほしい、など) を満たせなくなる。

我々は、このような問題に対応するため、探索初期

の途中時点で必要となったリソース量を計測し、最終的に探索全体で必要となるリソース量を予測して、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ったまま、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た[8]。

3.3 電力制御実験

本稿では、前述の分散探索アプリケーションを利用し、実際に消費電力量がどのように変化するかを測定して、消費電力削減を可能にする手法開発の基礎的な知見を得ることを目指した。

実験は、1 つの対局棋譜を用い、対局中に現れたそれぞれの局面で一定の深さまで探索するというワークロードを用いた。対局棋譜として、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」[⋆1] での対局結果を利用することとした。対局に現れた 86 局面を全て利用する。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 32: 常に 32 台を利用する場合
- optimal: 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、その必要台数で分散探索を開始するように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。)
- variable:浅い読みによる探索総時間予測[8] 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。optimal, variable 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おり、const 32 は実験開始時から終了時まで常に 32 ノードが電源 ON の状態である。

図 3 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。78 手目が最も探索リソースを必要とする局面であるが、const 32 と optimal が同程度の探索時間を示しており、optimal はこの局面に合わせて利用ノード数を制御していることがわかる。optimal はほぼ全ての局面で 60 秒から 120 秒の範囲で探索が終了しているが、const 32 は探索時間が短い局面と長い局面の差が大きい。const 32 はほとんどの局面でリソースを投入しすぎであった、と考えることもできる。optimal は最初の数手で探索時間が長くなり、5 手目では 180 秒近くを要している。これは、実験開始時に電源が入っていなかったノードを利用し始めようとしたため、サーバに電源を投入して OS の起動を待つ間、要求量より少ないノード数で探索を行わなければならなかった結果である。variable は必要なリソース量の予測が不正確な場合の制御例であり、optimal よりは探索時間が延びている。

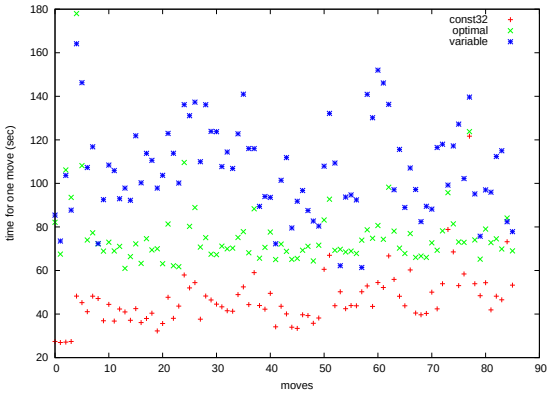

図 3 手毎の消費時間
Fig. 3 Execution time of each move

また、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 4** 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。計算ノードに関しては 1 節で述べた PDU により実際に実験中の消費電力を測定している。ネットワークスイッチに関しては、実験に利用するサーバをどのラックから選択するかによってネットワークトポロジが変化し、利用するスイッチの数も変化してしまうため、ここでは仮想的に、32 ノードの計算ノードが 16 ノードずつ 2 つのラックスイッチに収容されている環境を想定し、17 ノード以上を利用するときに限って 2 つめのネットワークスイッチの電源が ON になる、という制御が行われているとして消費電力を計算した。ネットワークスイッチは表 2 に示した事前実験結果から、1 台で 230 ワットの電力を消費するとした。図 4 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 32 は常に 32 ノードの計算ノードを利用しているため、使用電力量の変化は 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、4kW 程度の消費電力で稼働しているが、この時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。optimal、variable は総探索時間が const 32 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 4 の消費電力時間推移をもとに、計算期間中に使

⋆1 http://www.ipsj.or.jp/50anv/shogi/index2.html

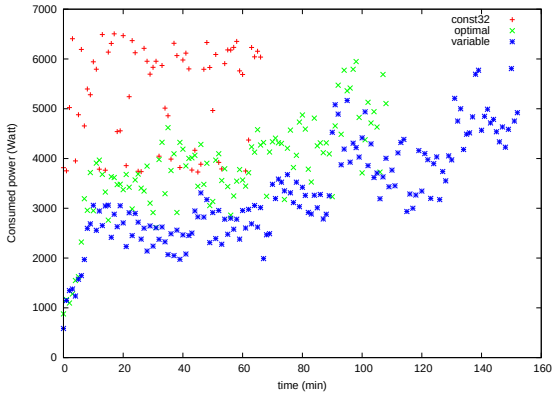

図 4 消費電力
Fig. 4 Power consumption

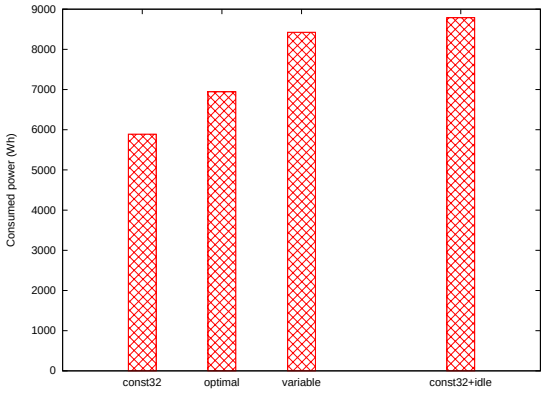

図 5 積算消費電力
Fig. 5 Power consumption (accumulated)

用した積算電力量を求めたものを**図 5** に示す。残念ながら、optimal の制御を行った場合の方が const 32 より多量の電力を使用する結果となっている。これは、optimal 制御の時に探索全体の所要時間が const 32 より延びたためであると考えられる。

この結果は、全ての計算要求 (本実験の場合、一局全ての局面) が最初に与えられている場合には、Elastic な制御を行わない const 32 の方が消費電力を削減できることを示している。しかし、もし計算要求が一度に与えられなかった場合、すなわち本実験で探索対象局面がある一定のペースで与えられるような場合を考えると、結果は変わってくる。計算要求の与えられるペースに比較して多すぎるリソースを投入している const 32 では、計算が終了した後の期間について、サーバが idle な状態で過ごさなければならない。今、仮に計算要求の与えられ方がちょうど optimal 方式で処理できる程度であったと仮定すると、const 32 は “optimal の処理時間 – const 32 の処理時間” の間だけ、計算サーバを idle 状態で動かし続けなければならなかったことになる。この場合の積算電力量を計算すると、図 5 の “const32 + idle” の系列になる。この場合、optimal より大量の電力を必要としており、Elastic な制御によって電力を削減する余地があったことがわかる。

4. 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。このようなプログラミングを可能にする処理系は、Elasticity を生かした消費電力の削減についても考慮しなければならない。電力の削減においては、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付帯設備も含めて電力制御が行えるようなシステムが必要となる。本稿では、そのようなシステムを設計実装し、実際のクラスタ上で稼働させた例を紹介した。また、動的な構成変更が行えるワークフロー処理系 GXP において、本電力制御システムと連動して Elastic な計算が行えるスケジューリング制御機能を追加し、コンピュータ将棋アプリケーションを題材に、制御を行わない場合と比較して消費電力がどのように変化するのかを測定した。結果、計算を行っている期間だけを考慮した場合は Elastic な計算制御による電力削減効果は得られなかったが、計算需要の与え方によっては電力削減の可能性があることが示された。

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り扱いなど、様々な改良が必要である。総使用電力量に制限を加えるなど、運用における制約条件の反映方法についても検討していく必要がある。また、バッチキューシステムとの連携、分散ストレージシステムとの連携など、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、双方に望まれる特性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。

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, *Communications of the ACM*, Vol.53, No.4, pp.50–58 (2010).
2) Barroso, L. A. and Hölzle, U.: The Case for Energy-Proportional Computing, *Computer*,

Vol.40, No.12, pp.33–37 (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, *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, ASPLOS XIII, pp.48–59 (2008).
4) Nishikawa, N., Nakano, M. and Kitsuregawa, M.: Energy Efficient Storage Management Cooperated with Large Data Intensive Applications, *28th IEEE International Conference on Data Engineering (IEEE ICDE 2012)* (2012).
5) Abts, D., Marty, M.R., Wells, P.M., Klausler, P. and Liu, H.: Energy proportional datacenter networks, *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pp.338–347 (2010).
6) 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).
7) 激指開発チーム：将棋プログラム「激指」. http://www.logos.ic.i.u-tokyo.ac.jp/~gekisashi/.
8) 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).