CPU使用率とメモリのプロセス間交換にもとづく動的なスループット向上手法

岡本太一[†] 前田敦司[†] 山口喜教[†]

計算の結果をメモリ上にキャッシュするプログラムやガベージコレクションを用いるプログラムのように，多くのメモリを使用することでスループットを向上できるプログラムは数多い．しかし，こうした時間と空間にトレードオフのあるプログラムを複数実行している状況において，それぞれのプログラムにどの程度のCPU使用率やメモリ使用量を割り振るのが最適かという問題の解は自明でない．

本研究では，ある仮定のもとでプロセス間でCPU使用率とメモリを交換することにより，双方のプロセスが共にスループットを向上できることを示す．またこの考え方に基づいてシステムのCPUとメモリの資源配分を動的に最適化するモデルを示し，その機構を提案する．この機構は個々のプロセスにスループットおよび資源割り当て状況の監視と，資源割り当ての変更を行うインターフェースを含む．さらに，プロセス外部に資源の適切な等価交換比率の算定と交換の仲介を行うデーモンを用いることにより，資源割り当てを自動的に最適化する．

Apache HTTP ServerとRedisというKey-value Storeに対する実装を提案し，議論を行う．

A Dynamic Throughput Improvement based on Inter-Process Trade of CPU Time and Memory

Taichi Okamoto,[†] Atusi Maeda[†] and Yoshinori Yamaguchi[†]

There are some class of programs that can improve their throughput by adding more memory; e.g. programs which cache computation results on memory, or programs using garbage collection. It is non-trivial to allocate memory optimally for multiple instances of programs that have time-space trade-off.

In this research, we show that, under certain assumption, it is possible for two processes to improve mutual throughput by trading their CPU consumption rate and memory pages. We also give a model and an implementation framework to dynamically optimize the systemwide allocation of CPU and memory. This framework includes an API to monitor throughput and resource allocation of each process. An external agent computes appropriate rate of resources and mediate trade, and thus optimize resource allocation automatically.

We implement this framework to Apache HTTP Server and Redis, a key-value store.

1. はじめに

Webサーバ計算機上においてはApache HTTP Server[1]をはじめとしてRedis[7]，MySQL[4]などの複数のアプリケーションが動作しているのが一般的である．どのプロセスにどれだけのCPU使用率やメモリ使用量を割り振れば，もっともWebサーバの性能が高くなるのだろうか．

現在，こうした資源の調節は管理者である人間が経験や勘を頼りに行っていることがほとんどである．たとえば急激なアクセスの増加に対しては，その都度Apache HTTP Serverの設定のチューニングを行なっている．こうした人間の関与が大きい状態は好ましいとはいえない．しかし自動化するとしても，一般にOSなどが外部からプロセスのスループットを観測するのは困難である．プロセスのCPU使用率が非常に高かったとしても，実行中の処理がアプリケーションのタスクなのか，ガベージコレクションなのかはプロセスしか知らない．つまり，自らのことをよくわかっているプロセス自身の関与が乏しい現状には，改良すべき点がある．

また，多くのプログラムにはメモリを占有しないヒューリスティクスが組み込まれている．たとえば，アプリケーションのスタート時には小さなサイズしか占有せず，処理が進行するにつれてだんだんと利用する量を拡大させるような機構をもっている．結果として，負荷がかかっているにもかかわらず，利用されて

† 筑波大学大学院 システム情報工学研究科
University of Tsukuba, Graduate School of Systems and Information Engineering

いないメモリがシステム内に存在しうる．もし空きメモリがあれば，それを有効に利用してスループットを向上すべきである．

計算機の処理能力あるいは計算機に存在する計算資源の量は，CPU 使用率のほかにもメモリ使用量やネットワークバンド幅，ディスク I/O など多数存在する．しかし，これらの資源のうちどの資源にどれだけの価値を置くかは利用するアプリケーションや利用される状況によって異なり，一般的な基準は存在しない．

本研究では，これらの計算資源のうち CPU 使用率とメモリ使用量の適切な比較尺度やスループットに与える影響が等価となる交換比率について考察する．またその等価交換比率をもとに，プロセスのスループットを最大化できる CPU 使用率とメモリ使用量の割り当ての組を自動的に探索する手法について議論する．この手法を用いて，各プロセスが限りある資源を有効に利用して最大限のスループットを出せるようにするメソッドの確立を目指す．また，プロセス間で資源を交換する一般的な資源管理のインターフェースについて展望する．

本研究では，時間と空間にトレードオフがあるプログラムを対象として考えている．たとえば，以下のような一定の処理が継続する環境をターゲットとして捉えている．

- Apache HTTP Server や nginx[5] といった Web サーバと Redis や memcached[3] などのキーバリューストアが協調して動作している環境
- 複数の仮想マシンをホストしている環境

特に本研究においては，Apache HTTP Server と Redis が動作している Web サーバにおける資源配分について考える．

2. 基本的なアイデア

基本的なアイデアを示す．

(1) CPU 使用率とメモリ使用量の 2 つの資源を交換できるようにする
(2) お互いのプロセスが得をできる，CPU 使用率とメモリ使用量の交換相手を探す
(3) 両者が CPU 使用率とメモリ使用量を交換する
(4) **双方のプロセスのスループットが向上**する

この処理をスループットを最適化したいプロセスに対して永続的に行うことで，スループットが最大化された状態を維持できる．

目指しているのは，**図 1** に×印で示した位置にあるスループットを，Efficient Frontier 曲線のどこかに近づけることである．この曲線は，双方のプロセスがお互いに相手のスループットを犠牲にせずに到達できるスループットの上限の集合を表す．この曲線に近づけば近づくほど，プロセスのスループットが向上できている．なお，この曲線のどこかには，システム全体のスループットの最高点がある．しかし，システム全体のスループットの定義自体も自明とはほど遠い複雑な問題である．特にデスクトップ PC などの汎用的な環境ではなおさら定義は困難である．

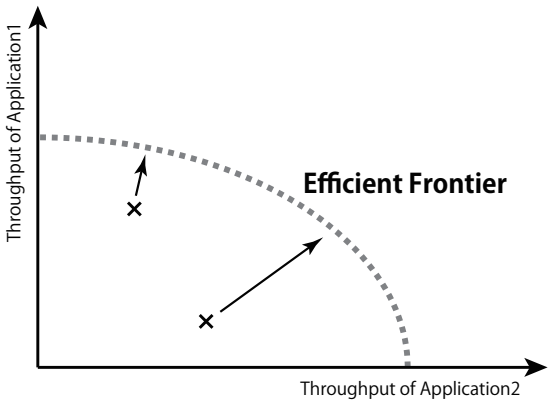

図 1 目指す方向

本研究では，他のプロセスの性能を犠牲にすることなく，あるプロセスのスループットを改善させる（パレート改善）ことを目指す．これ以上改善できない状態を（パレート）最適とし，この状態に到達させることを目標とする．

3. 関連する研究

関連する研究は大きく 2 つに分けられる．

- 対象が通信リソースである研究
- 対象が分散コンピューティング環境である研究

まず，対象が通信リソースである研究について述べる．これらの研究は文献 9)10) のように，インターネット・サービス・プロバイダのバックボーンなどに代表される複数のユーザの通信が同居する環境をターゲットにしていることが多い．そういった環境において，それぞれのユーザに対して効率的なリソースの割り当てを研究している．多くの研究では，通信帯域の獲得をオークション方式で行っている．

次に，対象が分散コンピューティング環境である研究について述べる．これらの研究は文献 8)11)12) のように，分散コンピュータ上のノード間での資源配分について取り扱っている．これらの研究で扱う資源は CPU やメモリにとどまらず，ネットワーク帯域やディスク容量にも及ぶ．また，処理するデータのロードバランシングやフロー制御の改良も視野に研究が行われ

ている．しかし，多くの研究ではノード間での資源の最適化や資源の交換を仲介するマーケットブローカについて扱っており，各ノード内における資源配分については詳しく取り扱っていないように思われる．多くの研究では資源の獲得を自由市場方式で行っている．

本研究は計算機内で資源の交換を行う．つまり，分散コンピューティングのグリッドにおける資源配分の前の段階として，各ノードで資源の逼迫度合いを測定可能である．グリッド上の各ノードどうしが資源を取引する上での取引比率は，まずノード内で決定されている方が望ましいと思われ，その点において本研究は有用であると考えている．

こうした関連研究をふまえ，本研究の位置づけを**表 1** にまとめた．

4. 提案の概要

4.1 時間と空間にトレードオフのあるプログラム

キャッシュを持っているプログラムに代表される，時間と空間にトレードオフのあるプログラムについて議論する．こうしたプログラムは実行時にメモリを多く割り当てることで，もしくは CPU 使用率を多く割り当てることで同じ出力を得るまでの実行時間が短くなる．すなわちスループットが向上する．

たとえば計算結果をキャッシュするプログラムは一般に，メモリを多く割り当てることによりキャッシュ量が増え，計算の回数が減りスループット向上が望める．CPU 使用率を多く利用することでも，同様の結果が得られる．またガベージコレクションを用いた処理系でプログラムを動作させる場合も，ヒープ領域に多くの仮想記憶領域を割り当てた方が処理時間が一般に短くなる．Apache HTTP Server などの Web サーバは，多くのメモリを利用してコネクションを処理する子プロセスを大量に生成することで，単位時間に処理できる HTTP 要求数が増大する．これもまた，スループットが向上しているといえる．

こうしたプログラムにおいて，CPU 使用率の増大によるスループット向上は**図 2** のように線形であると考えられる．それに対し，メモリ使用量の増大によるスループット向上は**図 3** のように一般に飽和する（上に凸な右上がりの曲線になる・増加の幅が減少する）と考えられる．本研究で対象とするプログラムはこうした特性をもつと仮定する．つまり，あるプロセスのスループットを $T(C, M)$（C は CPU 使用率，M はメモリ使用量）としたとき，式 1・2・3 が成り立つ．

$$\frac{\partial T}{\partial C} = A \ (A > 0) \tag{1}$$

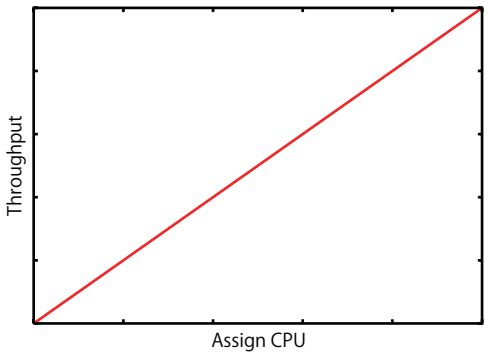

図 2 CPU 使用率の割り当てとスループットの関係

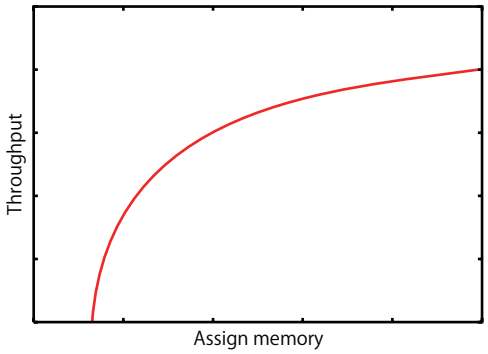

図 3 メモリ使用量の割り当てとスループットの関係

$$\frac{\partial T}{\partial M} > 0 \tag{2}$$

$$\frac{\partial^2 T}{\partial M^2} < 0 \tag{3}$$

こうした特徴のプログラムは，微小な範囲で CPU 使用率をメモリ使用量で代替できると考えられる．すなわち，メモリ使用量を減らすかわりに CPU 使用率を多く割り当てることで，プロセスのスループットが変化しない．逆も同様に，CPU 使用率の減少をメモリ使用量の割り当てで補える．

たとえば，CPU 使用率を 5%追加して得られる計算結果を，20MiB のメモリにキャッシュできるとする．ここで CPU 使用率を 5%減らしたとしても，メモリ使用量を 20MiB 増やせばスループットに変化は現れないと考えられる．すなわち，CPU 使用率を 5%とメモリ使用量 20MiB は交換可能といえる．

こうした時間と空間にトレードオフのあるプログラムでは，どれだけのメモリを使用できるかをユーザが外部からパラメータとして与えられるものが多い．必要に応じて利用するメモリ量を増加させていく作りのプログラムであっても，上限はやはりパラメータとして与えることができるものが多い．

表 1 本研究の位置づけ

	通信リソース	分散コンピューティング環境	本研究
対象	バックボーンネットワーク	グリッド内のノード	計算機内
取引する資源	ネットワーク帯域	ノード内のすべての資源	CPU とメモリ
最適化の方式	オークション方式が多い	自由市場方式が多い	自由市場方式

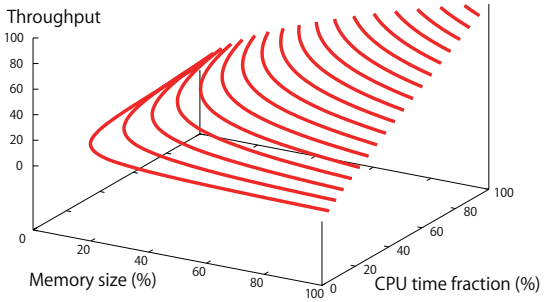

図 4 資源とスループットの関係 (3 次元)

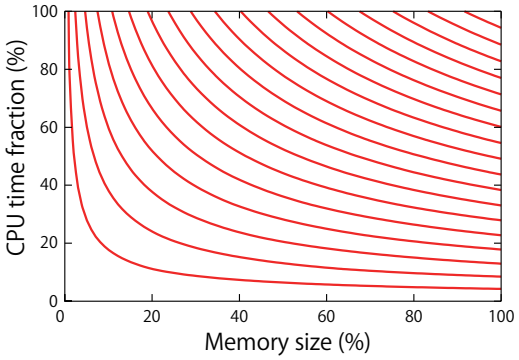

図 5 資源とスループットの関係 (2 次元)

4.2 CPU 使用率とメモリ使用量の交換

図 2・図 3 に示した CPU 使用率およびメモリ使用量の増加とスループット向上のグラフより，これらの資源とスループットの関係が**図 4** のように表せる．図 4 は Z 軸が増加するにつれて，スループットが向上している．図 4 を上から見たグラフを**図 5** に示す．図 5 は右上のほうがよりスループットが高い．

図 4・図 5 のグラフ上の曲線は，同じスループットが出せる資源配分（CPU 使用率とメモリ使用量の組）の集合である．また，ある点における曲線の接線の傾きは，**CPU 使用率とメモリ使用量のスループット上等価となる交換比率**を表している．図 5 のグラフから CPU 使用率とメモリ使用量が等価にスループットへ寄与する量がわかる．すなわち，スループットを指標として，CPU 使用率とメモリ使用量を比較・換算・交換できるようになった．

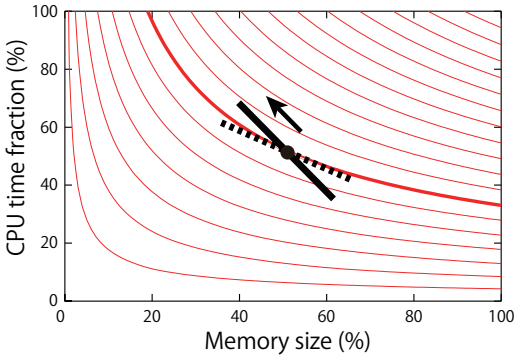

図 6 異なる等価交換比率によるスループット上昇

4.3 スループットが向上する理由

もし，あるプロセスの資源の等価交換比率と異なる比率で資源が交換できたとする．そのときは資源を交換することによって，**図 6** のようにより高いスループットの点へ遷移できる．

図 5 の資源とスループットの関係を表す曲線は，一般にプロセスごとに異なる．つまり，それぞれのプロセスの資源配分の座標における接線の傾き（資源の等価交換比率）も異なる．これは，プロセスが行っている処理や内部状態がそれぞれ異なるためである．

すなわち，資源の等価交換比率の異なるほかのプロセスと資源を交換することにより，プロセスのスループットを向上できる．たとえば，**図 7** のようにプロセス A・B の等価交換比率の中間の取引比率で資源を交換することにより，以下の 2 つのことがいえる．

- **双方のプロセスのスループットが向上**（パレート改善）
- プロセスどうしの等価交換比率の差が縮小

詳しくは 5.3 節も参照されたい．

もしプロセス間で資源の等価交換比率に差がない場合，資源交換は行われない．これは現時点がパレート最適であるということで，これ以上のスループット向上の余地はない．ただし，図 1 に示したようにパレート最適な状態は複数存在する．

4.4 近 傍 探 索

プロセスが自らの資源の等価交換比率（図 4 や図 5 に示した曲線）を既知とするならば，等価交換比率をもとにした最適な資源配分は容易である．しかし，実

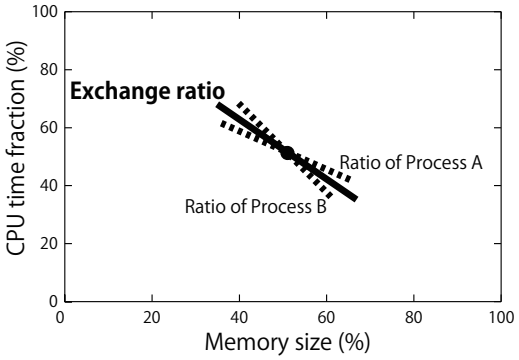

図 7 双方の傾きの中間の取引比率で資源を交換

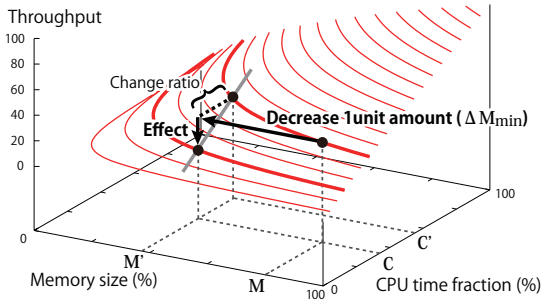

図 8 近傍探索による資源の等価交換比率の推定

際のプログラムでは処理が入力によってある程度の範囲内で変化し続けることが予想され，プロセスの内部状態は絶えず変動する．そのため，プロセスが必要とする CPU 使用率やメモリ使用量は一定ではなくなることを考えると，これは現実的な仮定とはいえない．

この解決策として，動的に等価交換比率を推定する近傍探索を提案する．本手法は現在の資源配分を微小な範囲で変更し，それによるスループットの変化を測定することで，現時点における資源の等価交換比率を求める．これは図 5 のグラフ全体を知るのではなく，現在の資源配分の座標における局所的な接線の傾きだけを求める手法である．

近傍探索は以下の手順で行う．

(1) メモリ使用量を単位量減らす（**図 8** の “Decrease 1unit amount”）
(2) 結果としてスループットに変化がおきる（図 8 の “Effect”）
(3) スループットは CPU 使用率に比例するという図 2 の仮定より，メモリを単位量減らした時に同じスループットを維持するのに必要な CPU 使用率がわかる
(4) メモリ使用量を減少させる前の CPU 使用率との差から，メモリ単位量と同等にスループットに寄与する CPU 使用率がわかる（図 8 の “Change ratio”）
(5) 資源の等価交換比率がわかる

詳しくは，5.1 節で述べる．

5. 資源交換モデル

5.1 資源の等価交換比率

資源の等価交換比率 P を**メモリ使用量あたりのスループット上等価な CPU 使用率**と定義する．C を CPU 使用率，M をメモリ使用量として P は式 4 のように書ける．

$$P = \frac{\partial C}{\partial M} \tag{4}$$

資源の等価交換比率は 4.4 節で示した近傍探索を用いて，メモリ単位量（ΔM_{min}）から離散的に近似する．ΔM_{min} については 6.5 節で詳述する．

プロセスのスループットを $T(C, M)$（C は CPU 使用率，M はメモリ使用量）とする．近傍探索する直前のスループットを $T(C, M)$ とする．ここからメモリ使用量を単位量減らして $M' = M - \Delta M_{min}$ とすると，スループットは $T(C, M')$ となる．メモリ使用量を単位量減らした状況において以前のスループットを維持できる CPU 使用率を C' とすると，$T(C, M) = T(C', M')$ である．これらの値は図 8 にも示した．

ここで図 2 の CPU 使用率とスループットの関係の仮定より，式 5 が導ける．

$$\frac{T(C', M')}{T(C, M')} = \frac{C'}{C} \tag{5}$$

以上より，等価交換比率 P を線形に近似した $\tilde{P}$ を式 6 とする．

$$\begin{aligned}\tilde{P} &= \frac{C' - C}{M - M'} \\ &= \frac{C' - C}{C} \times \frac{C}{M - M'} \\ &= \frac{T(C', M') - T(C, M')}{T(C, M')} \times \frac{C}{\Delta M_{min}} \\ &= \left(\frac{T(C, M) - T(C, M - \Delta M_{min})}{T(C, M - \Delta M_{min})} \right) \\ &\qquad \times \frac{C}{\Delta M_{min}}\end{aligned} \tag{6}$$

式 6 の括弧内はスループットの変化率である．なにをスループットとするかはアプリケーションごとに定義される．ただし，その指標はアプリケーションのスループットの変化を客観的に捉えている必要がある．

図 2・図 3 に示した資源とスループットの関係の仮定より，CPU 使用率を変化させずメモリ使用量を減少させると，必ず $T(C, M - \Delta M_{min}) < T(C, M)$ と

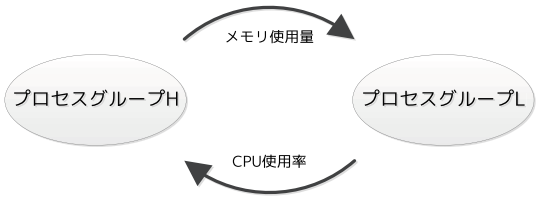

図 9 資源交換の概念図

スループットが減少する．よって，式 7 が成り立つ．

$$\frac{T(C,M)-T(C,M-\Delta M_{min})}{T(C,M-\Delta M_{min})} > 0 \qquad (7)$$

式 6 において $C > 0$ かつ $\Delta M_{min} > 0$ である．これと式 7 より，式 6 は必ず $\tilde{P} > 0$ である．

5.2 プロセスグループ

資源を交換する主体をプロセスグループと呼ぶ．これはアプリケーションの管理単位である．Apache HTTP Server のグループや Redis のグループのようなアプリケーションごとのまとまりと考えてもらってよい．プロセスグループどうしが資源を交換する．

グループには複数のプロセスが所属することもあるし，単体のプロセスしか所属しないこともある．たとえば，前者は Apache HTTP Server，後者は Redis があげられる．

5.3 資源の等価交換比率をもとにした資源交換

ある 2 つのプロセスグループ H とプロセスグループ L との間で資源を交換する状況を考える．この 2 つのグループの資源の等価交換比率をそれぞれ P_{higher} と P_{lower}（ただし $P_{higher} > P_{lower}$ ）とする．このとき，2 つのプロセスグループは**図 9** のように交換を行う．

- プロセスグループ H
  - メモリ使用量をグループ L に移譲する
  - CPU 使用率をグループ L から受領する
- プロセスグループ L
  - CPU 使用率をグループ H に移譲する
  - メモリ使用量をグループ H から受領する

自分より低い等価交換比率のプロセスグループがあれば，メモリ使用量を CPU 使用率に交換する．つまり，より低い等価交換比率のグループにメモリ使用量を差し出して CPU 使用率を受け取る（**図 10**）．逆に，自分より高い等価交換比率のプロセスグループがあれば，CPU 使用率をメモリ使用量に交換する．つまり，より高い等価交換比率のグループに CPU 使用率を差し出してメモリ使用量を受け取る（**図 11**）．

取引する際の取引比率は同じ値であるので，図 10 と図 11 の矢印の傾きは同じである．

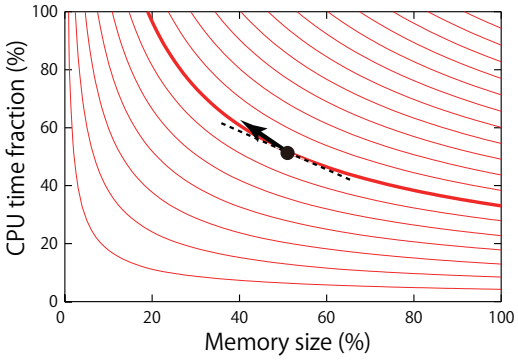

図 10 資源の等価交換比率の高いプロセスグループ

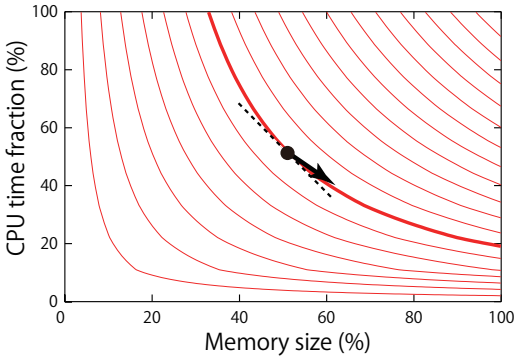

図 11 資源の等価交換比率の低いプロセスグループ

5.4 資源交換の際の取引比率

資源交換時においては，資源を交換するプロセスグループの間で，交換の基準となる資源の取引比率（取引レート）を決定する必要がある．この取引レート $P_{exchange}$ は，P_α・P_β を資源交換する 2 つのプロセスグループの等価交換比率として，式 8 とする．

$$P_{exchange} = \frac{P_\alpha + P_\beta}{2} \qquad (8)$$

現在は単純に両者の平均を取引レートとする．将来的にはプロセスグループに重みをつけて，どちらかのグループに偏った値にすることも考えられる．

5.5 一度に交換する資源の量

資源の交換は資源の等価交換比率が高いプロセスグループ H が等価交換比率が低いグループ L にメモリ使用量 ΔM を，プロセスグループ L がグループ H に CPU 使用率 ΔC を移譲することで行われる．一度の交換でプロセスグループどうしが交換する資源の量 ΔM と ΔC は，式 9 と以下に示す制約条件より決めることとする．なお，式 9 において ΔC_{min}・ΔM_{min} は資源の取引単位量である．

$$P_{exchange} \simeq \frac{\Delta C}{\Delta M}$$

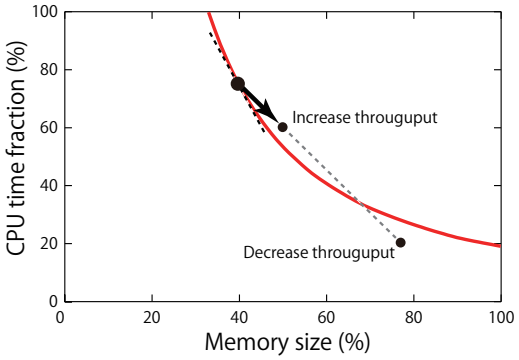

図 12 $\Delta C \cdot \Delta M$ の大きさによる問題

$$\simeq \frac{k_1 \Delta C_{min}}{k_2 \Delta M_{min}} \ (k_1, k_2 : \text{Integer}) \quad (9)$$

制約条件のもとに左辺を右辺で近似する．k_1, k_2 は取引単位量を $P_{exchange}$ に近づけるための取引係数である．取引係数は整数でなければならない．

制約条件を適用する際には，条件の順序によって得られる結果が変わってくることが考えられる．条件の優先順位を事前に決定しておく必要がある．

5.5.1 制約条件：$P_{exchange}$ との許容誤差

近似においては，許容相対誤差 d 以下の誤差しか許さない．この誤差は式 10 とする．

$$d \geq \left| P_{exchange} - \frac{k_1 \Delta C_{min}}{k_2 \Delta M_{min}} \right| \div P_{exchange} \quad (10)$$

この定数 d は，事前に外部から与える．

5.5.2 制約条件：取引係数の小ささ

取引係数 k_1, k_2 は，選択可能な値のうちできるだけ小さくなければならない．大きい値では交換する資源量が増大し，5.5.3 節で述べる問題が発生しうる．

ほかにも，$\Delta C \cdot \Delta M$ が充分小さくない場合に発生しうる問題として**図 12** がある．図 12 中の曲線は図 5 の曲線をひとつ取り出したものである．$\Delta C \cdot \Delta M$ が充分小さいならスループットが上昇する（“Increase throughput”の点）が，もし大きすぎる場合には交換後にスループットが低下（“Decrease throughput”の点）する可能性がある．

5.5.3 制約条件：交換の保証

$\Delta C \cdot \Delta M$ は式 11・12 を満たす．

$$\Delta C < C_{lower} \quad (11)$$

$$\Delta M < M_{higher} \quad (12)$$

ここで C_{lower} は資源の等価交換比率の低いプロセスグループ（CPU 使用率を移譲する側）の CPU 使用率，M_{higher} は等価交換比率の高いプロセスグループ（メモリ使用量を移譲する側）のメモリ使用量である．

もし式 11 を満たさない ΔC で取引すると，このグループは CPU 使用率について破産し，取引を続行できなくなってしまう．同様に式 12 では，これがメモリ使用量についていえる．

これはプロセスグループの等価交換比率が，現在の資源量を元に決められた値ではないために起こる．等価交換比率と現在の資源量は互いに独立しており，直接的な関係がない．等価交換比率は局所的なスループット変化と ΔM_{min} を元に決定されている．そのため，これらをもとに決定した $P_{exchange}$ で資源割当量に占める相対的な割合が大きい量を交換するのは適切ではない．5.5.2 節で述べた取引係数を小さくする必要性はここからきている．

5.6 プロセスグループのマッチング

すべてのプロセスグループの等価交換比率を，高い順にソートした配列を式 13 の P_{board} とする．

$$P_{board} = \{P_0, P_1, \ldots, P_n\} \ (P_i \geq P_{i+1}) \quad (13)$$

等価交換比率の差が大きいグループどうしが資源を交換したほうがスループット向上効果がある．P_{board} の中で最も差が大きいものは P_0 と P_n である．これらの等価交換比率のグループをマッチングし，資源を交換する（**図 13**）．以降 P_0 は $P_{highest}$，P_n は P_{lowest} と表記する．

図 13 で示したマッチングの例は，あくまで理想的なものである．実際の資源交換時には，メモリを相手に移譲するプロセスグループ H（$P_{highest}$ のグループ）が，交換に際してメモリを減少させることに伴うコスト c_m を資源の等価交換比率に加味しなければならない．メモリを減らす際には，解放する領域に存在するデータをファイナライズしたり他のページに退避させたりする必要が生じるためである．さらに，任意の量で資源を交換できるわけではないので，等価交換比率を量子化するときの量子化誤差を e も考慮する必要がある．この等価交換比率が理想からずれてしまうメカニズムを**図 14** に示す．メモリ減少コストはメモリ使用量を移譲する際にのみ発生するので，ここでは CPU 使用率を受領する場合にのみ影響する．

以上のコスト及び誤差をもとに，資源交換が発生するのは式 14 の場合である．

$$P_{highest} + e_{highest} > P_{lowest} + e_{lowest} + c_m \quad (14)$$

ところで図 14 では，左ほどメモリの等価交換比率が高いとも読める．また，CPU 使用率を受領することはメモリ使用量を移譲することである．逆にメモリ使用量を受領することは CPU 使用率を移譲することでもある．

5.7 資源交換モデルのまとめ

ここまで述べた資源交換モデルをまとめ，目的を整

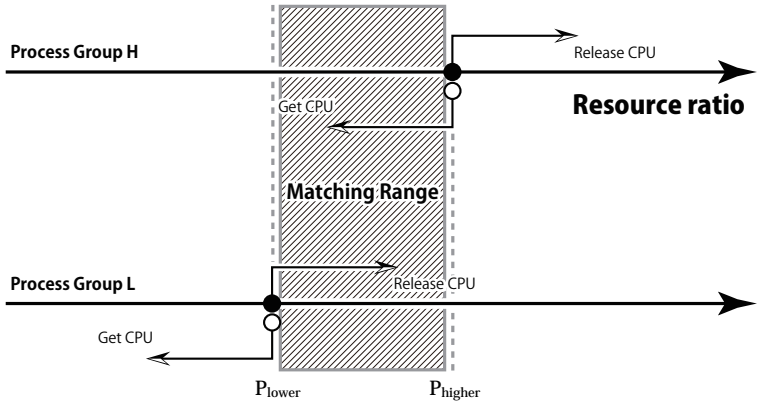

図 13 理想状態での等価交換比率マッチング

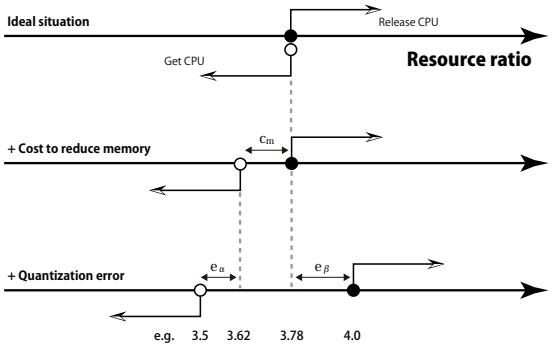

図 14 等価交換比率がずれるメカニズム

理する．本節では**表 2** のモデルを使って話をすすめる．この資源交換における資源量とスループットの変化は，スループットを $T(C, M)$（C は CPU 使用率，M はメモリ使用量）とすると**表 3** のようになる．

目指すのは，式 15・16 の成立である．

$$T_{higher}(C_{higher}, M_{higher}) \leq T_{higher}(C_{higher} + \Delta C, M_{higher} - \Delta M) \quad (15)$$

$$T_{lower}(C_{lower}, M_{lower}) \leq T_{lower}(C_{lower} - \Delta C, M_{lower} + \Delta M) \quad (16)$$

たとえ ΔC や ΔM が C_{higher} や M_{lower} にとって非常に小さくても，交換の意味はある．資源交換量がわずかでも，資源交換によりスループットが向上している．頻繁に資源を交換しないとすると，長期的に見れば十分なメリットがあると考えられる．

6. 実　　装

6.1 パラメータの取得方法

実装においては，CPU 使用率 C を Control Groups の `cpu.shares` の値から取得し，これを調整することで CPU 使用率を制御する．また，メモリ使用量 M はプロセスのワーキングセットの値とする．

Control Groups については，6.2 節で述べる．

6.2 CPU 使用率の変更方法

CPU 使用率の変更には，Linux カーネルの Control Groups（ cgroups ）を用いる．cgroups とはカーネル 2.6.24 にマージされたリソース割り当て管理用の機能である．著名なディストリビューションの現時点における最新版のほとんどに対応している．

Control Groups は当初は CPU だけの管理だったが，最近はメモリや一部の I/O も管理できるようになった．また，プロセス単体での制御だけでなく，あるプロセスから派生した子プロセスも自動的に親プロセスの資源管理を継承する．実際に，Apache HTTP Server はこの親プロセスと子プロセスのモデルで動作しており，こうしたアプリケーションの管理に簡単に利用できる．

この仕組みはカーネルがもっているため，すべてのアプリケーションに対して汎用的に利用可能である．

6.3 スループット関数 T

モデルで用いたプロセスグループ固有のスループット関数 T を，今回の実装では以下のように定義する．

Apache HTTP Server においては単位時間に処理できた HTTP 要求数とする．この値はアクセスログの行数を計測することで測定する．なお，1 アクセスにつき一行のログが出力される．

Redis はキャッシュヒット率とした．Redis は統計情報を内部に持っており，それを利用する．

6.4 CPU 単位量 ΔC_{min}

ΔC_{min} とは資源交換で交換する CPU 使用率の最低量（取引単位量）である．この値は Control Groups を用いて調整するのでリニアに変更できる．よって，CPU 使用率 1% を ΔC_{min} とする．

表 2 資源交換モデルの変数

	取引比率 or 交換量	等価交換比率の高い交換者 (H)	等価交換比率の低い交換者 (L)
取引比率 or 等価交換比率	$P_{exchange}$	P_{higher}	P_{lower}
CPU 使用率	ΔC	C_{higher}	C_{lower}
メモリ使用量	ΔM	M_{higher}	M_{lower}

表 3 資源交換モデル

	はじめ	近傍探索中	資源交換後 ($P_{higher} > P_{lower}$)
H	C_{higher}	C_{higher}	$C_{higher} + \Delta C$
	M_{higher}	$M_{higher} - \Delta M_{min}$	$M_{higher} - \Delta M$
	$T_{higher}(C_{higher}, M_{higher})$	$T_{higher}(C_{higher}, M_{higher} - \Delta M_{min})$	$T_{higher}(C_{higher} + \Delta C, M_{higher} - \Delta M)$
L	C_{lower}	C_{lower}	$C_{lower} - \Delta C$
	M_{lower}	$M_{lower} - \Delta M_{min}$	$M_{lower} + \Delta M$
	$T_{lower}(C_{lower}, M_{lower})$	$T_{lower}(C_{lower}, M_{lower} - \Delta M_{min})$	$T_{lower}(C_{lower} - \Delta C, M_{lower} + \Delta M)$

6.5 メモリ単位量 ΔM_{min}

ΔM_{min} とは資源交換で交換するメモリ使用量の最低量（取引単位量）である．これを決める方法はアプリケーションごとに異なる．これはアプリケーションの性質や設定，制御方法の違いによる．

6.5.1 直接メモリ使用量を調整できるアプリケーションの場合

直接メモリ使用量を調整できるアプリケーションの ΔM_{min} は式 17 とする．

$$\Delta M_{min} = 2^n \tag{17}$$

ここで 2^n とは，アプリケーションが動作している計算機のメモリページサイズである．メモリページサイズの値は，プロセッサのアーキテクチャやアドレッシングモード，OS などに依存している．一般に i386 や x86_64 では 4KiB である．

こうしたアプリケーションに Redis がある．Redis では `maxmemory` という値を変更することで，利用可能な最大実メモリサイズを直接調整できる．`maxmemory` の値は 1 バイト単位で任意に設定できるが，実際にはメモリページサイズを単位として調整する．

6.5.2 プロセス数を調節することで間接的にメモリ使用量を調整するアプリケーションの場合

プロセス数を調節することで間接的にメモリ使用量を調整するアプリケーションの ΔM_{min} は，式 18 とする．

$$\Delta M_{min} = \frac{M_{total}}{N_{process}} \tag{18}$$

ここで M_{total} とはプロセスグループ全体のメモリ使用量，$N_{process}$ とはグループのプロセス数である．すなわち，プロセスグループに所属している 1 プロセスあたりのメモリ使用量の平均が ΔM_{min} である．

この方法では ΔM_{min} に若干の誤差が生じるおそれがある．しかし，現時点ではプロセスごとのワーキングセットを一定と仮定し，プロセス数から実メモリサイズを間接的に制御することとする．

こうしたアプリケーションに Apache HTTP Server がある．Apache HTTP Server では `MaxClients` という値を調整することで，HTTP 要求を処理する子プロセスの最大生成数を変更してメモリ使用量を変化させる．Apache HTTP Server においては，クライアントからの要求に答えるプロセス 1 つ分のメモリ使用量が ΔM_{min} である．

6.5.3 それ以外のアプリケーションの場合

現時点においては，ここまでにあげたほかに分類されるアプリケーションは想定していない．

6.6 メモリ減少コスト c_m

モデルで c_m としていたメモリ減少コストは，アプリケーションごとの実装では以下のようになる．

- Apache HTTP Server：事実上 0
- Redis：キャッシュエントリのインデックスを再構築するのにかかるコスト

Apache HTTP Server のメモリ減少コストが事実上 0 なのは，メモリを減少させる方法によるものである．Apache HTTP Server では `MaxClients` を変更してメモリ使用量の増減を行うが，この設定を適用するのを `apachectl -k graceful` というコマンドで行う．この方法では，処理中のリクエストが完了しだい子プロセスを終了させるので，現在のリクエストが破棄されない．よって，スループットに対する影響は発生しないと考えられるためである．

7. 資源交換の枠組み

7.1 機能の概要

資源の交換に必要と考えている機能は 2 つある．

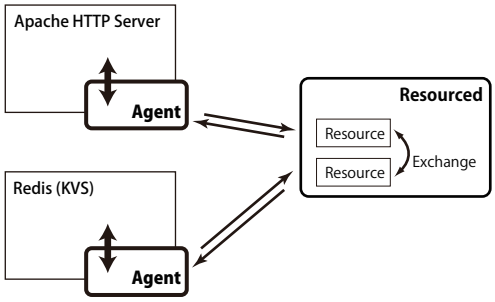

図 15 エージェントとリソースデーモンの実装例

- プローブの役割
- マッチメーカの役割

ひとつめのプローブの役割は，プロセスの資源配分を管理・監視し，また制御する機能である．プロセスに資源配分を強制したり，スループットの監視を行う．ほかにも定期的に近傍探索を行い，資源の等価交換比率を求める．この機能はアプリケーションごとに作成されるエージェントとして，もしくはアプリケーション内にサブルーチンとして実装するべきである．

これはアプリケーションにおいて，一般にスループットの指標は共通でないためである．また，OS などが外部からプログラムの振る舞いを観察することによってスループットを測定することは困難なためでもある．たとえば，時間のかかる計算の結果をキャッシュ（メモ化）するプログラムにおいて，資源配分を変えることで CPU 使用率を向上できたとする．しかし同時にキャッシュミスが大幅に増えているならば，スループットが向上しているとはいいがたい．すなわち，スループットはアプリケーション独自の方法で，アプリケーション自身もしくはアプリケーションをよく知る別のプログラムが計測・計算する必要がある．

ふたつめのマッチメーカの役割は，各プロセスのプローブから得た情報の一元管理を行い，それをもとにプロセス間の資源マッチングを行う機能である．この機能は計算機に対して 1 つあれば充分である．プローブからの決まった情報を決まった形でマッチングするだけなので，汎用性のあるリソースデーモンとして実装するべきである．

これらのプログラムの実装例を**図 15** に示す．図 15 は Apache HTTP Server と Redis に実装した場合の例である．エージェントは対象アプリケーションと通信しあう別のプログラムとして実装している．また，リソースデーモンが資源のマッチングのために存在している．

エージェントは資源の取引が行われない場合でも，定期的にメモリ使用量を増減させてスループットの変化を測定（近傍探索）する．そこで得られた資源の等価交換比率をリソースデーモンに対して報告する．リソースデーモンでは，各エージェントから報告のあった等価交換比率をマッチングする．もしマッチングに成功した場合は，資源の交換を行う．リソースデーモンは各プロセスが資源の交換後に使用できる資源の量を計算し，その値をエージェントに通告する．エージェントはリソースデーモンから示された値をもとに，担当しているアプリケーションの資源配分を変更する．

```
loop
  // デーモンからデータ受信
  data = recv()

  if data == false then
    // データを受信できていない
    mode = trigger() // 次の処理を選択

    if mode == ‘wait’ then
      // 待機
      isleep() // かしこく待つ
    else if mode == ‘localsearch’ then
      // 近傍探索
      ratio = search() // 近傍探索

      // 等価交換比率をデーモンへ送信
      send(ratio)
    endif
  else
    // データを受信できた
    enforcer(data) // 新しい資源配分を強制
    ratio = search() // 近傍探索

    // 等価交換比率をデーモンへ送信
    send(ratio)
  endif
endloop
```

図 16 エージェントの擬似コード

7.2 擬似コード

エージェントとリソースデーモンの擬似コードをそれぞれ**図 16** と**図 17** に示した．

エージェントとリソースデーモン間の通信にはノンブロッキングなプロセス間通信手段を用いる．たとえば，Ruby の `Queue` を利用することが考えられる．

`isleep` メソッドは TCP/IP の Binary Exponential Backoff と似たアルゴリズムを用いて，待機を行

```
bd = init() // 板を初期化

loop
  // エージェントからデータ受信
  while data = recv() do
    update(data, bd) // 板を更新
  endwhile

  if is_update_board() &&
                pair = match(bd) then
    // 板が更新されている
    // かつ資源のマッチングに成功
    vols = volume(pair, bd) // 交換量の決定

    // 新しい資源量を双方のエージェントに通告
    send(vols, pair)
  else
    // 板が更新されていないかマッチングに失敗
    isleep() // かしこく待つ
  endif
endloop
```

図 17 リソースデーモンの擬似コード

う．たとえば 2 秒待ってすぐまた isleep が呼ばれた場合 4 秒，さらには 8 秒のように待つ．

図 16 の search は 4.4 節の近傍探索を行い，enforcer では Control Groups やアプリケーションの設定を変更することで，実際にプロセスグループの資源量を調節する．図 17 の match では 5.6 節の資源マッチングを行い，volume では 5.5 節で述べた資源交換量を決める．

8. 考　　察

8.1 課　　題

本研究の課題として，近傍探索による資源の等価交換比率の推定精度やメモリ増減によるオーバーヘッド，割り当てたメモリ使用量と実際に使用しているメモリ使用量の乖離が考えられる．

近傍探索によって資源の等価交換比率をどれだけの精度で求められるのかは重要な問題である．理論的には最適な結果が得られるとしても，実際の等価交換比率の測定精度によっては性能向上が得られない可能性がある．たとえば，近傍探索中にスループットが変化していたとしても，そのスループットの変化が近傍探索によるものなのか，外部要因なのかを知ることはたいへん難しい．また，あるの数の倍数の値をとった場合にのみスループットが極端に良くもしくは悪くなるといった，特異点が存在する可能性も考えられる．こうした問題に対しては，近傍探索にサンプリングを併用するなどして，解決を図りたいと考えている．

さらに，近傍探索は動作中のプロセスのメモリ使用量を変化させるため，それに伴うオーバーヘッドが生じる．より高いスループットを出せる状態に遷移する際にも，メモリ使用量の減少を伴う場合がある．こちらも同様に，オーバーヘッドが顕在化してくる可能性がある．こうしたメモリ使用量の増減によるスループットへの影響については，アプリケーションごとにかなりの差がある．たとえば，6.6 節で述べた Apache HTTP Server においては，その値が事実上 0 であると考えられるのに対し，memcached は相当大きくなる．memcached では利用可能なメモリ使用量の上限を変更するのに再起動が必要であり，この再起動はキャッシュ中の全オブジェクトの喪失を意味する．こうしたアプリケーションへの対応も考えなければならない．

そして，アプリケーションにメモリ使用量を割り当てたとしても，その資源が使用されることが必ずしも保証されていない．もちろんこれはメモリ使用量だけに限らず，CPU 使用率でも同様に起こりうる問題である．逆に，割り当てた量以上にアプリケーションがメモリを占有することも考えられる．6.5 節で述べたように，Apache HTTP Server においても Redis においても，設定できるのはあくまでメモリ使用量の最大値である．たとえば，メモリを 500MiB 利用可能と設定していても，RSS の値は 300MiB 程度にとどまっているかもしれない．また，6.5.2 節で述べたアプリケーションの場合，ΔM_{min} の計測値に誤差があった場合に，割り当てた量を超えるメモリ使用が発生しうる．これは ΔM_{min} を実際のメモリ量より少なく見積もってしまった場合に起こる．

こうした割り当てたつもりの量と実際に使用している量の乖離は，定期的にこれらの差を監視し，使っていない資源を自動的に回収する機構を準備することで回避できると考えられる．可能な限り，これらの 2 つの値の差をこの機構が吸収することで，より効果的な資源配分が可能になるだろう．また，メモリを想定以上に利用してしまう問題については，6.2 節で触れた Control Groups を活用することで解決できる．6.2 節でも少し述べたが，cgroups は CPU 使用率の管理以外にメモリ使用量の管理もできる．この機能を利用することで，プロセスグループが利用できる実メモリサイズの上限を強制できる．これで，想定以上のメモリ占有が完全になくなるので，公平性が担保される．

8.2 今後の展望

今後の展望として，システム全体のスループットの最適化や，CPU 使用率やメモリ使用量以外の資源への応用が考えられる．

複数のアプリケーションが協調してひとつの問題を解くようなシステムにおいては，すべてのプロセスに平等に資源を分配することがシステム全体のスループットを向上させる上で最適とは限らない．たとえば Web アプリケーションがリクエストを処理するために RDBMS で検索を行い，その結果をキーバリューストアを用いて主記憶上にキャッシュするシステムを考える．このときそれぞれのアプリケーションに資源配分を行う上で，CPU 使用率に関してもメモリ使用量に関しても公平性はスループット向上に直結するとはいいがたい．データベースの処理が支配的ならば，RDBMS に CPU 使用率やメモリ使用量を多量に与えることがスループット向上に寄与するかもしれない．もしくは，キーバリューストアのヒット率が極めて高いならば，RDBMS のメモリ使用量を通常より減らしても，キーバリューストアに割り当てるメモリ使用量を増やした方がスループットの向上により寄与するかもしれない．こうした場合に，プロセスごとにどれだけの優先度を考慮していくか，またその計測も必要である．

また，CPU 使用率とメモリ使用量という 2 つだけの資源を扱うモデルから，ネットワークバンド幅や外部記憶との I/O バンド幅，ディスク容量など多数の資源を扱うモデルへの拡張が自然な方向として考えられる．ほかにも，メモリ使用量を割り当て可能資源としてではなく，電力を消費するデバイスとして見る考え方もある．スループットのほかにも電力という指標を導入するのである．メモリをむやみにアプリケーションに割り当てるのではなく，メモリの電源を落としたほうがコスト減につながるといった状況において，この指標は有効である可能性がある．ゆくゆくは，分散コンピューティングにおけるノード間での資源取引への拡張も期待できるだろう．

9. おわりに

本研究では，時間と空間にトレードオフのあるプログラムが複数存在するシステムにおける，CPU 使用率とメモリ使用量の最適な資源配分のモデルについて述べた．このモデルでは，個々のプログラムのスループットに対する寄与として等価となる，CPU 使用率とメモリ使用量の等価交換比率を用いて資源配分を決定する．また，この等価交換比率を用いて CPU 使用率とメモリ使用量を交換することにより，すべてのプロセスで資源配分の最適化を図るインターフェースを提案した．

この資源配分の最適化は以下の流れで行われる．

(1) 近傍探索して資源の等価交換比率を推定
(2) 等価交換比率をもとに，交換できる相手を探す
(3) 資源を交換する
(4) お互いのプロセスのスループットが向上する

今後の展望として資源交換を行うエージェントとリソースデーモンの実装，資源の初期配分も含めたシステム全体での最適化，CPU 使用率とメモリ使用量以外の資源への応用などが考えられる．

謝　辞

本研究の一部は科学研究費補助金 挑戦的萌芽研究 課題番号 23650011 の補助を得て行われた．

参 考 文 献

1) The Apache HTTP Server Project, https://httpd.apache.org/.
2) Free VMware vSphere Hypervisor: Bare Metal Hypervisor (Based on VMware ESXi), https://www.vmware.com/products/vsphere-hypervisor/overview.html.
3) memcached - a distributed memory object caching system, http://memcached.org/.
4) MySQL :: The world's most popular open source database, https://www.mysql.com/.
5) nginx news, http://nginx.org/.
6) PHP: Hypertext Preprocessor, http://www.php.net/.
7) Redis, http://redis.io/.
8) Ferguson, D. F.: The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms (1989).
9) Semret, N.: Market Mechanisms for Network Resource Sharing, Technical report (1999).
10) 高橋英士: 市場メカニズムに基づく通信リソース配分法の研究, 博士論文, 早稲田大学大学院 (2003).
11) 松本尚: 市場メカニズムに基づく計算資源割り当て方式, 「さきがけ研究 21 」研究報告会「情報と知」領域講演要旨集 Dec, pp. 157–165 (2001).
12) 玉井森彦, 柴田直樹, 安本慶一, 伊藤実: PC グリッド環境での市場原理に基づいた資源共有方式, 情報処理学会論文誌, Vol. 47, No. 2, pp. 455–464 (2006).
13) 前田敦司: 計算資源の配分に関するプログラム間のインターフェース, 夏のプログラミング・シンポジウム報告書, pp. 59–66 (2009).