Vol.2022-OS-155 No.2
2022/5/26

サーバ基盤におけるハードウェア省電力設定活用のためのリソース管理方法

藤田 勝美[1,a)]　岩佐 絵里子[1,b)]　金子 雅志[1,c)]

概要：サーバの省電力化に有効な設定として CPU 動作周波数や電圧を負荷に応じて制御する DVFS がある．しかし消費電力と性能はトレードオフの関係にあり，省電力設定を有効にしたサーバを用いて安定したサービス提供をすることは性能面で課題がある．本研究では異なるポリシーでコンポーネント配置制御される仮想化されたアプリケーションが同一サーバ基盤上でサービスを提供する場合において，各アプリケーション要件に適した設定がされたサーバ群を選択しスケジューラへ提案するリソース管理機能を含むシステムを提案する．提案の有効性を示すための第一ステップとしてソフトウェアコンポーネントの動的配置シミュレーションコードを実装し評価結果から DVFS 設定が有効化されているサーバ基盤上でも性能優先アプリケーションを動作させ，性能劣化を抑制する効果があることを確認した．

キーワード：サーバ基盤，仮想化技術，性能保証，省電力

1. はじめに

1.1 背景

ビッグデータやクラウドコンピューティング産業の普及に伴い，データセンタの消費電力増加が問題となっている．データセンタの消費電力問題に対するサーバ基盤技術領域における解決策として仮想化技術を用いたシステム集約 [1] や Dynamic Voltage and Frequency (DVFS) を利用した CPU 消費電力制御 [2] などがある．仮想化技術では，ひとつの物理サーバ上に複数台の仮想マシンもしくはコンテナ (以降，コンポーネントとまとめて表記) を搭載し異なる処理を行うため，同一サーバ基盤上に様々なアプリケーション (App) で様々なサービスを提供することが可能である．その特徴を利用し，低負荷状態が続く時間帯にコンポーネント数を削減し使用するサーバ台数を削減するアプローチが仮想化技術を用いたシステム集約である．DVFS は負荷に応じて CPU コアの電圧や動作周波数を動的に制御することでサーバの消費電力削減を実現する技術であり，CPU の機能として搭載されている．DVFS 機能を有効にする場合はサーバの BIOS もしくは OS から設定が可能である．

しかし性能保証の観点から性能を優先した設定でサーバを運用しているケースがある．近年のデータセンタの消費電力増加や環境問題への意識の高まりを踏まえると，性能保証と同様に消費電力も運用における重要な課題であり消費電力削減に有効な技術は積極的に活用することが望ましい．

1.2 課題

サービス提供時に DVFS 機能を活用しようと考えた場合，異なるアプリケーションが混在する環境下で各アプリケーションが各々の要件を満たしつつサービス提供できることが課題となる．

本研究で対象とする環境は異なる種類のサービスを複数同時に提供するサーバ基盤とした．このような環境下で複数のスケジューリングアルゴリズムにより各サービスの目的達成 (最適化問題における目的関数の最大化や最小化など) のためにコンポーネント配置制御が行われる．このとき，サーバ基盤側の DVFS 設定を有効にすると App 側の性能要件とのミスマッチが発生することが懸念される．具体的に発生する問題として，DVFS 設定を有効にしたサーバ基盤上に性能要件の厳しい App を搭載した際に，DVFS 機能により低負荷時の CPU 動作周波数が低下することで処理性能が一時的に下がり，性能劣化を招く可能性がある．

2. 目的

サーバ基盤上で動作する App はサービスの内容により最適化の目的が異なるため，どのような環境下でも最適な

[1] NTT ネットワークイノベーションセンタ
[a)] katsumi.fujita.ze@hco.ntt.co.jp
[b)] eriko.iwasa.xt@hco.ntt.co.jp
[c)] masashi.kaneko.dr@hco.ntt.co.jp

スケジューリングを実現できる必要がある．そのためには異なる設定がされているサーバ群に対する適切な「使い分け」と「管理」の方法が必要とされ，この課題解決を本研究の目的とする．本研究では，異なる最適化目的を持つスケジューリングアルゴリズムが混在するサービス基盤上で各アルゴリズムがそれぞれの目的を達成し，かつ基盤全体のリソース利用効率を向上させるために物理・仮想リソース利用量やサーバの電源・設定情報などを管理・制御するリソース管理アルゴリズムを含んだシステムを提案する．

2 章以降の構成として 3 章では関連研究を紹介し，4 章では提案するリソース管理方法，5 章でシミュレーションによる評価方法を説明する．その後 6 章・7 章で結果と考察，8 章でまとめを述べる．

3. 関連研究

まず既存のコンポーネント最適配置制御の研究に関して説明する．コンポーネント最適配置制御は静的配置制御と動的配置制御の 2 つのスキームに分けられる [3]．静的配置制御とはサービス提供開始時のデプロイメントや運用中にサーバ基盤上に配置されているコンポーネントの配置を最適化する．一方，動的配置制御とは運用中の負荷増減や消費電力増減などの変化に対してコンポーネントのスケーリングやマイグレーションを行うことで動的に配置を最適化する．

最適化の目的関数もコンポーネント間の通信遅延のような性能に関する値やサーバ消費電力など多岐にわたる．さらに，一つの目的関数を最適化する単目的最適化問題もあれば，複数の目的関数を最適化する多目的最適化問題としても取り扱われている．数ある先行研究の中でも本研究と同様にサーバの省電力設定に着目しその設定を制御に含めた研究は以下のものが挙げられる．

Laszewski et al. [4] では DVFS 動的制御可能なクラスタのための VM 配置アルゴリズムを提案している．VM のデプロイメントの要求が到着すると，処理能力が高い計算ノードに優先的にデプロイしていくことでリソース利用効率を向上させ省電力化を実現する．Chuang et al. [5] では CPU 使用率に応じて負荷の高いマシンの VM をアイドルマシンへマイグレーションさせることでリソース利用効率を向上させ，エネルギー消費を削減するアルゴリズムを提案している．Patel and Bheda [6] は Chuang et al. [5] のアプローチを参考にしており，エネルギー消費だけでなく，処理の実行時間という性能指標を考慮に入れたライブマイグレーションアルゴリズムを提案した．

ただし，これらの先行研究が提案するような DVFS 設定を制御に含めたアルゴリズムと性能要件の厳しい App のための制御アルゴリズムを同一サーバ基盤上に搭載された個々のアプリケーションに適用し，サービス提供を行う場合，1.2 節で挙げた性能劣化が懸念される．そこで本研究では省電力設定の活用を目指し，コンポーネント最適配置アルゴリズムに依存しないリソース管理機能の検討を行った．

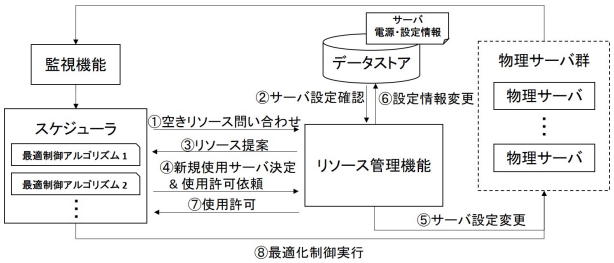

図 1 システム構成図

4. 提案

4.1 システム全体像

図 1 に提案するリソース管理機能を含むシステム構成図を示す．監視機能部とスケジューラ，リソース管理機能部とデータストアで構成され，それらが連携して仮想化された App が搭載される物理サーバ群の監視や制御を行う．監視機能部がコンピュート群の状態を監視し，スケジューラが新規サービス開始時のデプロイや運用中のサービスに対してマイグレーションやスケーリングといったコンポーネント配置制御を行う．スケジューリングアルゴリズムは App ごとの要件 (最適化の目的) に適したアルゴリズムが用意され，使い分けられる．リソース管理部では物理リソース使用量やサーバの電源・設定情報などを管理し，スケジューラからのリソース要求に対し，App 要件を満たすリソースを提供する役割を果たす．

具体的には，スケジューラによる最適配置推定計算時，App ごとに「最適化の目的」と「サーバの設定」にミスマッチが無いよう，全サーバの中から適切な設定のサーバ群を配置先候補とし，空きリソースなどの情報をスケジューラに伝える．この時，サーバの優先度を定め，スケジューラに渡す空きリソース情報について段階的に制限をかける機能を持たせる．したがってスケジューラでは実際に利用可能なサーバより少ないサーバ群内での最適配置推定と制御が行われる場合がある．以上がシステムと各機能についての概要である．次節ではリソース管理部の機能をより詳しく説明する．

4.2 リソース管理機能

コンポーネント最適配置スケジューラがデプロイメントやマイグレーション，スケーリング等の制御の必要性を判断し制御を実行する過程で，スケジューラは使用可能なリソースをリソース管理部に問い合わせ，それを受けたリソース管理部が App 要件に応じたリソースを選択し，スケジューラに提案することでスケジューラはリソースを把握する (図 1 中の 1 から 3 に対応)．

リソース管理部のリソース選択アルゴリズムを Algo-

Algorithm 1 リソース管理機能

Require: Resource requirement from scheduler
Ensure: resouce candidate list
1: $List \leftarrow 0$
2: $SchedulerType \leftarrow performance\_aware$ or $energy\_aware$
3: $C \leftarrow Resource\_requirement$
4: **if** $SchedulerType = performance\_aware$ **then**
5: $List \leftarrow Performance(List,\ C)$
6: **end if**
7: **if** $SchedulerType = energy\_aware$ **then**
8: $List \leftarrow Energy\_aware(List,\ C)$
9: **end if**
10: **return** $List$
11:
12: **function** PERFORMANCE($List,\ C$)
13: $List \leftarrow GetList(performance, powerON)$
14: **if** $Listsize < C$ **then**
15: $List.add(GetList(energy\_saving, powerON))$
16: **end if**
17: **if** $Listsize < C$ **then**
18: $List.add(GetList(performance, powerOFF))$
19: **end if**
20: **if** $Listsize < C$ **then**
21: $List.add(GetList(energy\_saving, powerOFF))$
22: **end if**
23: **return** $List$
24: **end function**
25:
26: **function** ENERGY_AWARE($List$)
27: $List \leftarrow GetList(energy\_saving, powerON)$
28: **if** $Listsize < C$ **then**
29: $List.add(GetList(energy\_saving, powerOFF))$
30: **end if**
31: **if** $Listsize < C$ **then**
32: $List.add(GetList(performance, powerOFF))$
33: **end if**
34: **return** $List$
35: **end function**
36:
37: **function** GETLIST($setting,\ power$)
38: $s \leftarrow setting$, $p \leftarrow power$
39: $n \leftarrow 0$
40: $list \leftarrow 0$
41: **for all** $n \leftarrow$ Server total number **do**
42: **if** server n setting $= s$ & server n power status $= p$ **then**
43: $list.add(n)$
44: **end if**
45: **end for**
46: **return** $list$
47: **end function**

rithm1 に示す．はじめにリソース管理部はスケジューラからリソース問い合わせを受ける．このとき，スケジューラの最適化の目的と必要なリソース量の情報を得る．今回，最適化の目的は性能優先と省エネ優先の 2 種類とした．そしてデータセンタ内の全物理サーバから最適化の目的に適した設定のサーバリストを作成する．作成したサーバリストの空きリソースの合計がスケジューラのリソース要求量以上である場合，リストをスケジューラに提供する．リソースが足りない場合はあらかじめ設定した優先度 (表 1, 2) に基づき，リソース要求量を超えるまでサーバをリストに追加する．サーバの優先度はデータセンタ内の全サーバをサーバ省電力設定の有無とサーバの起動状況により分類し，各サーバ群に優先度が付けられる．本研究ではサーバの省電力設定は performance mode と energy-saving mode の 2 パターンとし，それぞれは DVFS 機能 OFF, ON に対応する．

表 1 性能優先スケジューラのためのリソース優先度

HW 設定	Power ON	Power OFF
Performance mode	1	2
Energy-saving mode	3	4

表 2 省エネ優先スケジューラのためのリソース優先度

HW 設定	Power ON	Power OFF
Performance mode	採用しない	3
Energy-saving mode	1	2

リソース管理部から利用可能なサーバ群を提案されたスケジューラは与えられた範囲の中で最適な制御方法を推定する．推定結果から新規コンポーネントの追加や既存コンポーネントを別サーバへ配置変更が必要と判断し対象となるサーバが決定した場合，スケジューラはリソース管理部へサーバの使用許可を問い合わせる．問い合わせを受けたリソース管理部は対象サーバを使用可能な状態にするため，電源や設定が対象 App を搭載するのに適しているかを確認し，適切な制御（電源 ON や設定変更）を行った後，スケジューラへ使用許可を出す (図 1 中の 4 から 7 に対応)．このリソース管理部の処理を図 2 に示す．

本章で説明したリソース管理部を評価するためにシミュレーションを行った．次章ではシミュレーションモデルと評価方法を説明する．

5. シミュレーション

提案するリソース管理方法の有用性を判断するため同一データセンタ上のサーバを共有する複数の異なるアプリケーションがそれぞれの最適化の目的に応じてコンポーネントを制御するような状況で時間経過とともにコンポーネントの増減や配置，サーバ設定変更などがサーバの消費電力や使用率等に与える影響をシミュレーションで再現することが必要であり，コンポーネント動的配置シミュレーションコードを実装することとした．

5.1 シミュレーション概要

まず状況設定としてデータセンタ内に 2 種類の App が搭載され，異なるサービスを提供しているとした．一方は性能が求められるリアルタイム処理を行う App(以降「性能優先 App」と表記する) であり，もう一方はバッチ処理

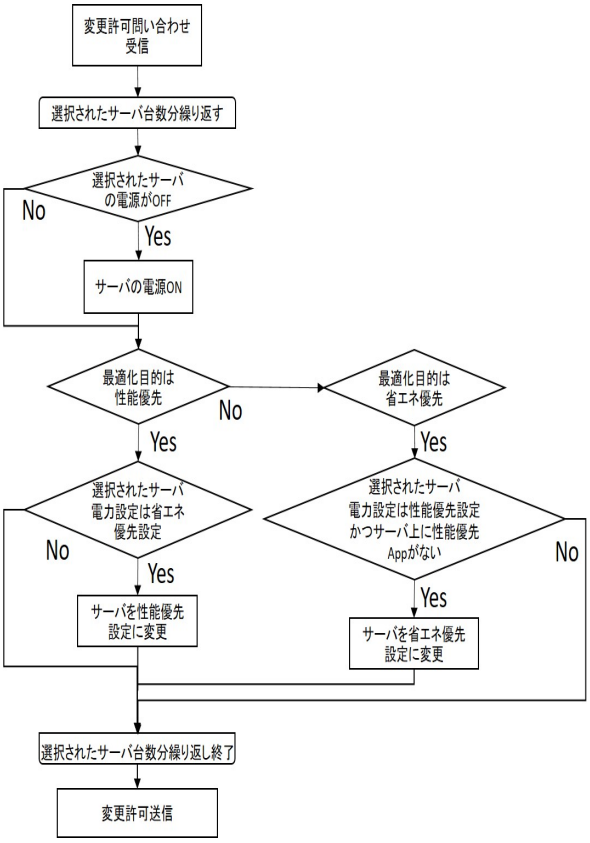

図 2 利用サーバ決定後のリソース管理部の処理

を行う App(以降「省エネ優先 App」と表記する) であるため性能は前者ほど求められないが消費電力削減が求められるものとした．そしてこの状況設定でデータセンタ内のサーバ稼働状況やコンポーネント増減などの変化を時間経過とともに追うことができるようなシミュレーションの検討を行った．シミュレーションコードは仮想マシン動的配置シミュレータである CloudSim [7] を参考に実装を行った．CloudSim ではサーバの計算処理能力を million of instructions per second (MIPS) で定義しタスク処理やサーバ使用率の計算に用いている．本研究のシミュレーションコードでもこの考え方を採用した．以下で計算内容の詳細を示す．

5.2 シミュレーションセットアップ

まず初期値として以下の値を設定した．

- シミュレーション時間 $T_{total} = 21600$ 秒
- 1 タイムステップあたりの時間 $T_{step} = 1$ 秒
- サーバ
  - サーバ台数 N_{HW} =20 台 (シミュレーション開始時点で稼働状態にあるサーバは 4 台とする)
  - シャットダウンまたは起動にかかる時 $T_{on\_off} = 60$ 秒
  - サーバ省電力設定は performance mode (DVFS 機能 OFF) と energy-saving mode (DVFS 機能 ON) の 2 パターンとする
  - 最大処理能力 (performance mode) $hw\_mips\_max\_perf = 3500$ MIPS
  - 最大処理能力 (energy-saving mode) $hw\_mips\_max\_energy = 3000$ MIPS
- 処理コンポーネント
  - 最大処理能力 (performance mode) $cp\_mips\_max\_perf = 700$ MIPS
  - 最大処理能力 (energy-saving mode) $cp\_mips\_max\_energy = 600$ MIPS
  - スケールアウトにかかる時間 $T_{scale\_out} = 60$ 秒
  - デプロイメントにかかる時間 $T_{deploy} = 60$ 秒
- 性能優先 App
  - 初期デプロイコンポ―ネント数 $N_{cp} = 7$ 台
- 省エネ優先 App
  - バッチ処理完了期限 $T_{deadline} = 900$ 秒
- 負荷量
  - 性能優先 App 最大負荷量 $W_{perf} = 35000$ MIPS
  - 省エネ優先 App 最大負荷量 $W_{enrgy} = 600 * 6 * 900$ MIPS

負荷量についてはタイムステップごとに到着する量を初期値として設定した．図 3 に設定した負荷量グラフを示す．性能優先 App では Google Traffic Transparency Report [8] の日本時間 2021 年 12 月 14 日午前 9 時 30 分から 24 時間分の web search traffic データを負荷として採用し，省エネ優先 App では一定時間ごとに負荷が到着するよう設定した．図 3 の縦軸は日本のトラヒックと全世界のトラヒック量の割合にある定数をかけたものが示されている．図 3 の横軸は Google web search traffic のデータ数を示しており，30 分刻みのデータが 48 個 (24 時間分) ある．今回のシミュレーション時間は 21600 秒 (6 時間)，1 タイムステップの計算で経過する時間を 1 秒と設定しているため，シミュレーション時間を 48 分割し，450 秒 (=21600/48) ごとに各タイムステップで到着する負荷量を変更するようにした．

また，本シミュレーションでのスケジューリングアルゴリズムにおける制御方法としてコンポーネント配置を性能指標を閾値としてコンポーネント数を増減設する方法 (水平スケーリング) を採用した．詳細は次節で説明する．

5.3 シミュレーション方法

シミュレーションでは 1 タイムステップ中で STEP0 から STEP6 までの計算を行うことでタスクの処理や使用率計算，コンポーネントの配置変更などの計算を行い，それを繰り返すことで時間経過による計算結果の変化を確認する．以下で各 STEP ごとの計算内容の詳細を説明する．

STEP0 負荷到着

性能優先 App の場合，図 3 の負荷量をパーセンテー

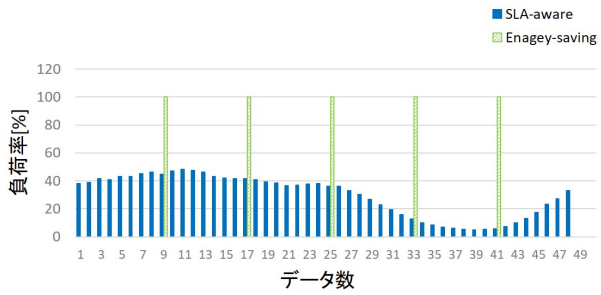

図 3 App の負荷量．青色 (塗りつぶし) 棒グラフが性能優先 App への負荷，緑色 (網掛け) 棒グラフが省エネ優先 App への負荷を示す．

ジで表される割合と見なし，W_{perf} を掛けることで単位を % から MIPS へ変換する．あるタイムステップ時刻 t で到着した性能優先 App の負荷量を w_t^p とする．シミュレーション開始時点であらかじめコンポーネントをサーバ上で起動させた状態にしておき，負荷 w_t^p が到着すると，データセンタ上で稼働する性能優先 App 用コンポーネントすべてに均等に割り当てる．この時，j 番目のコンポーネントに追加されたタスク w_t^p ともともと積まれていたタスクの合計を Q_{CPj} で表す．

省エネ優先 App の場合性能優先 App の場合と同様到着した負荷に W_{energy} を掛けることで単位を % から MIPS へ変換する．あるタイムステップ時刻 t で到着した省エネ優先 App の負荷量を w_t^e とする．省エネ優先 App の場合，シミュレーション開始時点ではコンポーネントを初期デプロイしないため，到着した負荷に対して制限時間内に処理が完了するために必要なリソース量 C_{cp}(省エネ優先 App コンポーネント数) を以下の式で算出し，得られた台数分のコンポーネントをサーバ上にデプロイした後，各コンポーネントに負荷を均等に割り当てる．

$C_{cp} = w_t^e/(cp\_mips\_max\_energy \times T_{deadline})$

ただしデプロイにかかる時間 T_{deploy} だけタイムステップが進んだ後からタスク処理を開始する．またデプロイ時の配置方法として，あらかじめ番号付けたサーバ群のなかから，空きリソースがあり，かつ番号が小さい順から詰めてく First Fit アルゴリズムを採用した．

STEP1 コンポーネント・サーバ使用率計算

性能優先・省エネ優先 App 共通で，あるタイムステップ時刻 t での j 番目のコンポーネントの使用率 U_{CPi} は以下のような式で求める．

$U_{CPi} = Q_{CPj}/cp\_mips\_max\_perf$

上式はサーバの設定が performance mode の場合の使用率を示しているが，energy-saving mode の場合 $cp\_mips\_max\_perf$ を $cp\_mips\_max\_energy$ に変更し使用率の計算を行う．

またあるタイムステップ時刻 t での i 番目のサーバの使用率 U_{HWi} は以下のような式で求める．

$U_{HWi} = \sum_j Q_{CPj}/hw\_mips\_max\_perf$

$\sum_j Q_{CPj}$ は i 番目のサーバ上で稼働するすべてのコンポーネントに積まれたタスクの合計値を示す．コンポーネントの使用率計算と同様，サーバの設定が，energy-saving mode の場合 $hw\_mips\_max\_perf$ を $hw\_mips\_max\_energy$ に変更し使用率の計算を行う．

STEP2 サーバ消費電力計算

サーバの消費電力モデルとして Goyal et al.[9] で定式化されている CPU 使用率を変数とする線形モデルを採用した．CPU 使用率をサーバの使用率 U_{HWi} と置き換え，i 番目のサーバの消費電力 P_{HWi} を以下の式で与える．

$P_{HWi} = aU_{HWi} + b$　(a, b 定数)

今回のシミレーションではラックマウント型サーバの使用を想定し，サーバの使用率増加に伴う消費電力増加より待機電力が大きい数値となるよう，サーバの設定ごとに定数 a, b を決定した．

performance mode:　$P_{HWi} = 10U_{HWi} + 100$

energy-saving mode: $P_{HWi} = 5U_{HWi} + 95$

STEP3 制御

性能優先 App の場合，サーバの使用率 U_{HWi} が 80% 以上もしくはコンポーネントに積まれたタスク Q_{CPj} がそのコンポーネントの最大処理能力の 2 倍以上だった場合を SLA 違反と見なしスケールアウト実行の判断基準とした．そして稼働中のサーバの中で使用率が最も小さいものスケールアウト先のサーバとして予約する．稼働中のサーバだけでは空きリソースが足りない場合は電源が OFF のサーバを選択する．スケールアウト完了までにかかる時間 T_{scale_out} だけタイムステップが進んだ後，選択したサーバにコンポーネントを追加する．

一方 SLA 違反がなく，稼働中のコンポーネントの使用率 U_{CPi} が 50%を下回るものがあった場合をスケールイン実行の判断基準とする．稼働中のサーバの内，最も使用率が高いサーバを選択し，そのサーバ上に搭載されているコンポーネントのなかで最も番号が小さいものをスケールインの対象とし，そのコンポーネントが保持するタスクをすべて処理し終えたタイミングでサーバ上から削除する．

省エネ優先 App の場合，省エネ優先 App の場合はスケーリングはマイグレーションなどの制御は行わず，与えられたタスクの処理が完了した時点でコンポーネントをサーバから削除する．

STEP4 シャットダウン

性能優先・省エネ優先 App 共通で，あるタイムステッ

プにおいて，サーバ上にコンポーネントが存在せず，アイドル状態であった場合，そのサーバをシャットダウンする．

STEP5 タスク処理

性能優先 App の場合コンポーネントに積まれたタスク Q_{CPj} から 1 タイムステップで処理される分のタスクを差し引く．

$Q_{CPj}^{after} = Q_{CPj}^{before} - U_{CPj} \times cp\_mips\_max\_perf$

省エネ優先 App の場合も上式の $cp\_mips\_max\_perf$ を $cp\_mips\_max\_energy$ にすることで同様の計算を行う．

STEP6 時刻更新

初期値として定義されたタイムステップあたりの時間 T_{step} でタイムステップの時刻を更新する．

5.4 評価方法

4 章で提案するリソース管理方法をソフトウェアコンポーネント動的配置シミュレーションで評価を行った．5 章で説明したシミュレーションモデルで時間経過に伴う状態変化を追った結果 (「提案リソース管理適用なし」と表記) とそのシミュレーションモデルの中に 4 章で説明したリソース管理アルゴリズムを組み込み時間経過に伴う状態変化を追った結果 (「提案リソース管理手法適用あり」と表記) を比較することで提案の有用性を評価する．提案リソース管理手法適用なしの場合は 4 章で提案した手法を組み込まずにシミュレーションを行う．従って，リソース管理部がスケジューラからリソース問い合わせを受けた際に提案するサーバ群はデータセンタ内に存在するすべてのサーバが候補となる．

評価項目は以下の 2 項目とする．

(1) App-物理サーバ設定のミスマッチ数

(2) サーバ全体の消費電力量

1 番目の項目は App の要件に対してサーバの設定が適切かどうかを確認する．1.2 節で説明したように，性能要件の厳しい App が energy-saving mode のような動作周波数が低くなる設定がされたサーバ上でサービスを提供する場合，性能劣化の可能性がある．1 番目の項目ではそのようなミスマッチが App-物理サーバ間で発生した回数を確認する．2 番目の項目はタイムステップごとに全物理サーバで消費される電力の総和確認し，提案リソース管理手法適用あり・なしで比較する．

また，5.2 節よりサーバ省電力設定は performance mode か energy-saving mode のいづれかとし，データセンタ内の全物理サーバのシミュレーション開始時点での初期設定を 3 パターン用意し初期条件の違いによるリソース管理効果の比較も行った．初期設定 1 パターン目はすべての物理サーバ設定が performance mode，2 つ目はすべて物理サーバ設定が energy-saving mode，3 つ目はサーバに着けた番号の偶数番号が performance mode，奇数番号が energy-saving mode とした．

表 3 SLA 違反率

初期設定パターン	1	2	3
リソース管理手法適用なし	27%	34%	36%
リソース管理手法適用あり	21%	22%	27%

表 4 消費電力量

初期設定パターン	1	2	3
リソース管理手法適用なし	20886 kJ	22584 kJ	23693 kJ
リソース管理手法適用あり	21548 kJ	21534 kJ	25812 kJ

6. 結果

シミュレーション結果を図 4 に示す．図 4 は評価項目 1 の App-物理サーバ設定のミスマッチ数の時間経過に伴う変化を示しており，左から物理サーバ初期設定パターン 1, 2, 3 の順に並べている．図 4 からは，サーバの初期設定がすべて performance mode である場合，提案リソース管理手法適用ありなしに関わらず物理サーバ上で処理を行うコンポーネントすべてにおいて App-物理サーバ設定のミスマッチがない結果となった．一方でサーバの初期設定がすべて energy-saving mode，もしくは performance mode と energy-saving mode 設定サーバが混在する場合，提案リソース管理手法適用なしの場合で App-物理サーバ設定のミスマッチが発生し，提案リソース管理手法適用ありでは初期段階では最大 4 台のコンポーネントの App-物理サーバ設定のミスマッチが確認できたものの，時間経過と共にミスマッチが解消される結果となった．

全シミュレーション時間において，性能優先 App が SLA 違反した時間の割合を求めた結果を表 3 に示す．今回のシミュレーションにおける SLA 違反の定義は 5.3 節の計算方法 STEP3 で説明した通りであるが，提案するリソース管理手法適用ありの場合の方がすべての初期設定パターンで SLA 違反率が小さいという結果になった．これらの結果から DVFS 使用環境で提案するリソース管理手法を適用することで CPU 動作周波数低下の影響を受けて性能要件の厳しい App の処理が劣化することを抑制する効果があると考えられる．

全シミュレーション時間において消費された電力の総和を求めた結果を表 4 に示す．単位はキロジュール (kJ) で示しており，初期設定パターン 2 以外でリソース管理手法適用ありの方が消費電力量が高い結果となった．

7. 考察

「App-物理サーバ設定のミスマッチ数」と「サーバ全体の消費電力量」の 2 つの評価観点を基に提案するリソース管理手法の効果の確認をシミュレーションで行った．提案するリソース管理手法は DVFS 機能を利用する状況下で課題としていた性能要件の厳しい App の性能劣化を防ぐこ

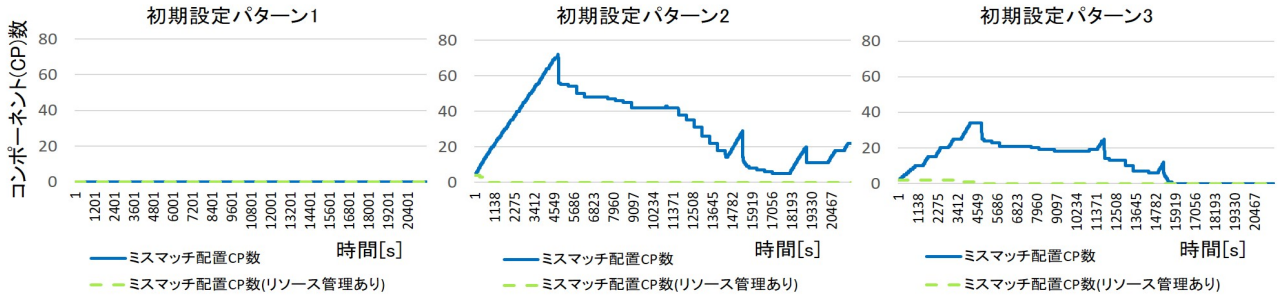

図 4 App-物理サーバ設定のミスマッチ数の時間変化

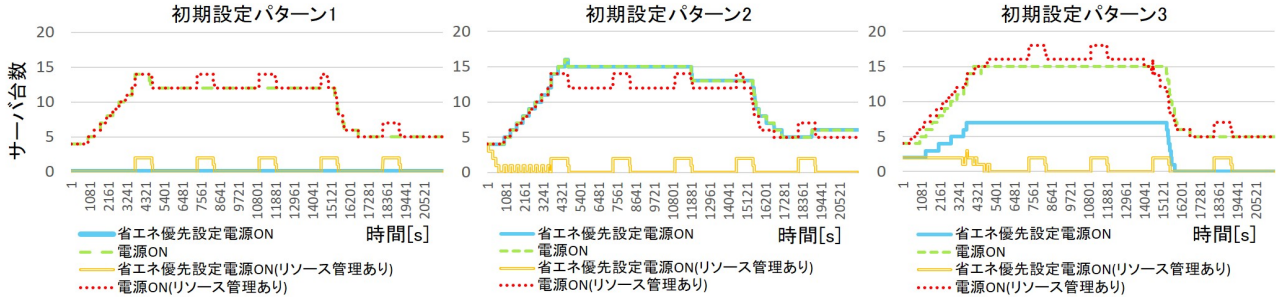

図 5 サーバ稼働台数の時間変化

とができる可能性があることはシミュレーション結果から示すことができた．一方でサーバ群全体で消費される電力量が増加するため省電力化の効果が期待できないことが示された．

初期設定パターン 1 と 3 でリソース管理手法を適用した場合に消費電力量が増加した理由としては，省エネ優先 App への負荷が到着した際に稼働するサーバの台数が増加したことが原因と考えられる．稼働中，つまり電源が ON の状態にあるサーバ台数の時間変化を図 5 に示す．図 5 の水色実線とオレンジ二重線がそれぞれリソース管理手法適応なし・ありの場合の電源 ON かつ energy-saving mode 設定のサーバ台数を示しており，リソース管理手法適用ありのオレンジ二重線は省エネ優先 App への負荷が到着したタイミングで台数が増加していることがわかる．それにより，電源 ON のサーバ台数 (performance mode, energy-saving mode 両設定を含む) に差が生じたことで消費電力量の差につながったと考えられる．これは提案するアルゴリズムで省エネ優先スケジューラのためのリソース優先度を表 2 のように設定したことが影響していると思われる．特に今回のシミュレーションでは，省エネ優先 App のコンポーネント配置を First Fit アルゴリズムを用いて行ったため，リソース管理手法適用なしの方が空きリソースへ効率よくコンポーネントを詰め込むことでサーバ全体の消費電力が削減され，DVFS 設定による電力削減効果よりも高い効果を示したと考えられる．しかし，初期配置パターン 2 では，図 5 からリソース管理手法適応ありの電源 ON 状態のサーバ台数 (赤色点線) がリソース管理手法適応なしの電源 ON 状態のサーバ台数 (黄緑色破線) を下回る期間が長いこと，そして表 4 から消費電力量が少ないことが確認できた．これはリソース管理手法を適応し，性能優先 App のためにサーバ設定を peformance mode にした際，そのサーバ上に存在する省エネ優先 App の処理性能も向上し，処理完了までの時間が短縮されたことでサーバ上にコンポーネントが存在する時間が短縮された結果コンポーネント集約率が上がり，サーバの稼働台数が削減されたことが理由であると考えられる．

ただし，表 2 で設定した優先度ルールにおいて省エネ優先 App を peformance mode 設定のサーバ上に性能優先 App と共に配置することはリソース競合による性能優先 App の性能劣化を招く可能性もあるため，性能優先 App の性能とサーバ全体の消費電力はトレードオフの関係にある．性能と消費電力のどちらを優先するか，またどこで折り合いを付けるかなどの検討の為には具体的な実サービスを想定した検証方法を考える必要がある．

今回のシミュレーションではラックマウント型サーバの使用を想定していたため，待機電力が大きく，サーバ設定の違いによる電力削減効果の幅が待機電力に対して小さくなるような電力モデルを作成しシミュレーションを行った．サーバ設定の違いによる電力削減効果の幅とサーバの待機電力との差が小さい機種の使用を想定した場合，省電力効果の結果が今回の結果と異なる可能性があり，様々な電力特性を持つサーバについてシミュレーションを行うことで，本提案を適用するにあたり最も効果を発揮するサーバの特徴を選定するという検討ができると考える．また，シミュレーションの精度を高めるために実際のサーバ消費電力データを基にした電力モデルを使用することも必要で

あると考える．

8. まとめ

異なるポリシーでコンポーネント配置制御される仮想化されたアプリケーションが同一サーバ基盤上でサービスを提供する状況下において，サーバ省電力設定 (DVFS 設定) を活用することを目指し，各アプリケーションに適した設定のサーバ群を選択しスケジューラへ提案するリソース管理方法を提案した．

提案する手法の有効性を確認するため，時間経過にともなうソフトウェアコンポーネント配置やサーバの設定・電源の変化をシミュレーションし，提案手法の適用あり・なしで性能劣化や消費電力量などの比較を行った．シミュレーションでは，サーバ省電力設定を活用したサーバ基盤上でも性能要件の厳しい App を搭載し，性能劣化を抑制しつつ運用することが可能であることが示されたが，性能が優先される分だけ消費電力はトレードオフの関係となり，省電力効果が見込めないという結果となった．

さらなる評価精度の向上に向けてサーバの消費電力や起動時間，コンポーネントのデプロイメントにかかる時間などを実機で測定し，その測定値を用いて計算をすることが必要であると考える．加えてコンポーネントのスケジューリングアルゴリズムも既存のサービスで利用されているものや，先行研究で提案されているものなど様々なバリエーションでシミュレーションを行い提案手法を評価しつつ，さらなる改善に取り組むことを今後の課題とする．

参考文献

[1] Rocha, I., Göttel, C., Felber, P., Pasin, M., Rouvoy, R. and Schiavoni, V.: Heats: Heterogeneity-and energy-aware task-based scheduling, *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, IEEE, pp. 400–405 (2019).

[2] Kuehn, P. J. and Mashaly, M.: DVFS-power management and performance engineering of data center server clusters, *2019 15th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, IEEE, pp. 91–98 (2019).

[3] Masdari, M., Nabavi, S. S. and Ahmadi, V.: An overview of virtual machine placement schemes in cloud computing, *Journal of Network and Computer Applications*, Vol. 66, pp. 106–127 (2016).

[4] Von Laszewski, G., Wang, L., Younge, A. J. and He, X.: Power-aware scheduling of virtual machines in dvfs-enabled clusters, *2009 IEEE International Conference on Cluster Computing and Workshops*, IEEE, pp. 1–10 (2009).

[5] Chuang, H.-S., Lee, L.-T., Chang, C.-Y. and Tseng, C.-Y.: Modified Packing Algorithm for Dynamic Energy-Saving in Cloud Computing Servers, *Journal of Electronic Science and Technology*, Vol. 11, No. 2, pp. 124–131 (2013).

[6] Patel, V. J. and Bheda, H. A.: Reducing energy consumption with DVFS for real-time services in cloud computing, *IOSR J (IOSR J Comput Eng)*, Vol. 16, No. 3, pp. 53–57 (2014).

[7] Buyya, R., Ranjan, R. and Calheiros, R. N.: Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities, *2009 international conference on high performance computing & simulation*, IEEE, pp. 1–11 (2009).

[8] Google: Traffic Transparency Report, Google (online), available from ⟨https://transparencyreport.google.com/traffic/overview⟩ (accessed 2022-04-12).

[9] Goyal, S., Bhushan, S., Kumar, Y., Rana, A. u. H. S., Bhutta, M. R., Ijaz, M. F. and Son, Y.: An optimized framework for energy-resource allocation in a cloud environment based on the whale optimization algorithm, *Sensors*, Vol. 21, No. 5, p. 1583 (2021).