令和 6 年度

修士学位論文

可用性とレスポンスタイムを考慮した VM の再配置手法

VM Relocation Method Considering Availability and Response Time

1275099　　柏原星司

指導教員　　横山和俊

2025 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

可用性とレスポンスタイムを考慮したVMの再配置手法

柏原星司

近年, クラウドコンピューティングが急速に普及している. クラウド事業者は安定したサービスを提供するにあったってSLAとして稼働率を保証する場合が多い. また, クラウドアプリケーション提供者はSLOというサービスが満たすべき目標を設定する場合が多く, SLOにおいてレスポンスタイムが設定される場合が多い. このように可用性を満たしながらレスポンスタイムを最小化するような手法が求められている. 菊森らは、可用性を考慮しながら電力の最小化を目指す動的配置手法について検討している. この研究において, ホストの高負荷による電力上昇について考えられている. しかし, 高負荷によるレスポンスタイムの上昇については考慮されておらず, レスポンスタイムについての評価も行われていない.

そこで, 本研究では可用性を考慮しながら, レスポンスタイムの最小化を目指す再配置手法について提案し, 有用性を評価する.

キーワード　クラウドコンピューティング, 仮想化, デュプレックスシステム, 待ち行列理論

Abstract

VM Relocation Method Considering Availability and Response Time

Joji KASHIWABARA

In recent years, cloud computing has been spreading rapidly. In order to provide stable services, cloud service providers often guarantee availability as SLA. In addition, cloud application providers often set SLOs, which are goals to be met by their services, and response times are often set in the SLOs. Thus, there is a need for a method to minimize the response time while satisfying the availability. Kikumori et al. have studied a dynamic placement method that aims to minimize power consumption while considering availability. In this method, power increase due to high host loads is considered. However, they do not consider the increase in response time due to high host loads, nor do they evaluate the response time.

Therefore, in this study, we propose a relocation method that aims to minimize the response time while considering availability, and evaluate its usefulness.

key words Cloud Computing, Virtualization, Duplex Systems, Queueing Theory

目次

図目次

表目次

第 1 章

はじめに

近年, クラウドコンピューティングが急速に普及している. 総務省が発行する令和 6 年度情報白書 [1] によると世界のパブリッククラウドサービスへの支出額は 2023 年に 5,636 億ドルまで増加すると見込まれており, 今後も市場の拡大が予想される. クラウド事業者は安定したサービスを提供するにあたって Service Level Agreement(以降 SLA と略す) として稼働率を保証する場合が多い. 実際に AWS の EC2 においては 99.99%の月間稼働率で利用可能であるように保証している [2].

また近年, クラウドアプリケーション提供者は Service Level Objective(以降 SLO と略す) というサービスが満たすべき目標を設定する機会が増加している. 具体的にある決済サービスの SLO を抜粋したものである [3].

- Availability
  - 1 時間測定して、99.99%のリクエストが、正常な結果を返す
- Latency
  - 1 時間測定して、 99%のリクエストが 100msec 以内に返る
- Quality
  - Sentry などで 10 分測定して、想定外のエラーが 0.1%

一般に上記に示すようにエラー率やレスポンスタイムを目標として設定する場合が多く, 得にリアルタイム性が求められるような決済サービスにおいてレスポンスタイムの向上が求められる.

菊森ら [4] は、可用性を考慮しながら電力の最小化を目指す動的配置手法について検討し

ている．この研究において, ホストの高負荷による電力上昇について考えられている. しかし, 高負荷によるレスポンスタイムの上昇については考慮されておらず, レスポンスタイムについての評価も行われていない.

そこで, 本研究では可用性を考慮しながら, レスポンスタイムの最小化を目指す再配置手法について提案する.

第 2 章

関連研究

クラウドコンピューティングでの, 仮想マシン配置手法として様々なものが検討されている. 例えば, 文献 [5] では，消費電力と SLA 違反率（以降 SLAV と略す）を考慮し，動的な負荷変動に対応した仮想マシンの配置アルゴリズムを提案している．消費電力の削減に対応した配置アルゴリズムと SLAV の削減に対応した配置アルゴリズムの 2 種類を用意し，その 2 つを選択的に適応させる提案手法を提案している．しかしながらここでは SLA を違反した際の可用性について考慮されていない.

SLA を違反した場合の可用性を考慮した配置手法として参考文献 [4] の手法が挙げられる．この論文では, 可用性と消費電力を考慮した配置手法について考えられている. 具体的には可用性を満たすような以下の 2 つの制約条件を設けることによって, 可用性を満たしながら省電力化を目指す配置手法について提案されている.

1. サービスの現用系と予備系を同じ配置にしない
2. 違うサービスの現用系と予備系の配置と同じ配置にしない

結果として, 現用系と予備系が 1:1 の場合において, 障害発生時間を抑えながら, 電力の削減が達成されていた.

また, レスポンスタイムを評価するにあたって CPU や I/O の負荷との相関関係について調査がなされている. 文献 [6] においてクラウド環境を対象とした WEB システムについて CPU 使用率とレスポンスタイムに正の相関関係があることが述べられており, レスポンスタイムへ強い影響があることが分かっている. 同様に I/O についても文献 [7] では, SSD におけるランダム read の読み込み量とレスポンスタイムとの相関について述べられており, ま

た文献 [8] においても, ランダムアクセスの IOPS とレスポンスタイムと相関があると述べられている.

レスポンスタイムのモデル化については, 文献 [9] において, クラウド環境でレスポンスタイムを評価するにあたって待ち行列理論の M/M モデルを用いることの効果について述べられている. その結果として M/M モデルを用いることによってレスポンスタイムに与える影響の特定に効果があることがわかっている.

第 3 章

システムモデル

本研究で想定するクラウド環境のモデルについて説明する.

3.1 可用性モデル

本研究では可用性モデルとしてデュプレックスシステムを想定する. デュプレックスシステムシステムにおいて一般に現用系と予備系が N 対 1 の関係で配置される場合が多い. しかし, 本研究においては 1 つの現用系に対して 1 つの予備系が存在するような 1 対 1 構成のデュプレックスシステムを前提として考える. また待機方式についても待機の状態からホットスタンバイ, コールドスタンバイ等いくつかに分けることができる. ここでは一定期間で同期をとるようなホットスタンバイで待機されているシステムについて考える.

3.2 レスポンスタイムモデル

複数の VM から 1 つのホストへリクエストが送られるようなクラウドシステムにおいて, M/M/1 モデルを用いることによってレスポンスタイムを近似できることが知られている [10]. 本研究では CPU, I/O の到着率がレスポンスタイムに影響を与えるものとして考える. 待ち行列理論の M/M/1 モデルにおいてレスポンスタイムは単位時間当たりのリクエストの平均到着率 (λ), 単位時間当たりの処理性能 (μ) を用いて以下の式で近似できることが知られている.

$$ResponseTime = \frac{1}{\mu - \lambda}$$

これは以下のように計算される負荷率 (ρ) に依存する.

$$\rho = \frac{\lambda}{\mu}$$

ここで, CPU と I/O がレスポンスタイムへ与える影響の大きさの比率 (以降影響度とする) が a, b としてレスポンスタイムを以下のようにモデル化する.

$$ResponseTime = \frac{a}{\mu_{CPU} - \lambda_{CPU}} + \frac{b}{\mu_{I/O} - \lambda_{I/O}}$$

ここで本研究では同機種クラスタを想定して考える. そのため, すべてのホストのサービス率 μ_{CPU}, $\mu_{I/O}$ は等しいものとして考える. またすべての VM の CPU, I/O の到着率は既知であるとして考える.

3.3　システム構成

システム構成を以下の図 3.1 に示す. ここではホスト ID を 1, 2, … , HOSTS として, サービス ID を A, B, …として示している.

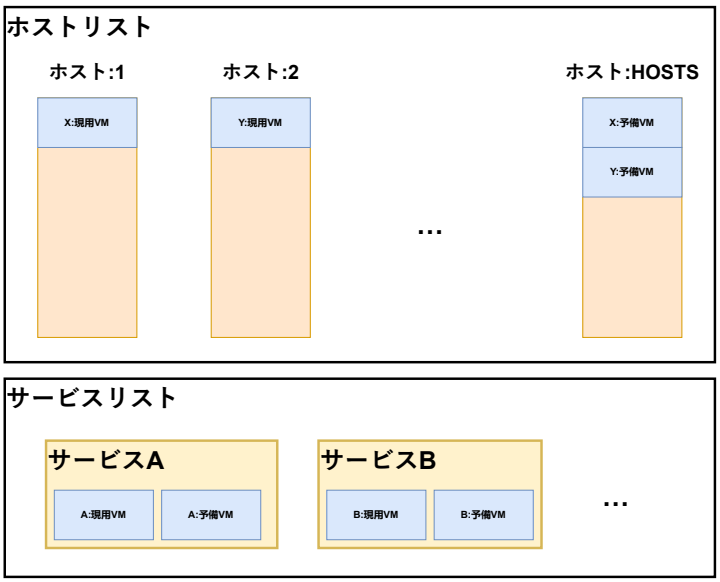

図 3.1　システム構成像

次にシステムの構成要素がそれぞれ保有する情報を以下の表に示す. ここで各 ID はサービス, VM, ホストが一意に決まる識別子である.

表 3.1　サービスが保有する情報

情報	説明
サービス ID	サービスの ID
現用系 VMID	現用系の VM の ID
予備系 VMID	予備系の VM の ID
現用系の到着率	現用系 VM のリクエスト平均到着率
予備系の到着率	予備系 VM のリクエスト平均到着率

表 3.2　ホストが保有する情報

情報	説明
ホスト ID	ホストの ID
サービス率	ホストが単位時間に処理できるリクエスト数
配置されている VM リスト	ホストに配置されている VM のリスト
ホストの総到着率	現在ホストが抱えている到着率の総和

表 3.3　VM が保有する情報

情報	説明
VMID	VM の ID
サービス ID	属するサービスの ID
配置先ホスト	VM が配置されているホストの ID

第 4 章

提案手法

既存研究において負荷上昇による電力の上昇については考慮されていたものの, レスポンスタイムの増加については考慮されておらず, その評価もなされていなかった. 本研究では複数の VM が 1 つの物理ホストにリクエストを与えるような M/M/1 モデルを採用し, レスポンスタイムと相関があることが分かっている CPU, I/O を M/M/1 モデルによってモデル化する. また, 文献 [4] における可用性条件を用いることによってレスポンスタイムと可用性を考慮した再配置を目指す.

4.1 目的関数

レスポンスタイム R を最小化する:

$$Minimize\, R = \sum_{j=1}^{m} \frac{a}{\mu_{CPU,j} - \sum_{i=1}^{n} \left(x_{ij} \cdot \lambda_{CPU,i}^{\text{primary}} + y_{ij} \cdot \lambda_{CPU,i}^{\text{standby}} \right)} + \frac{b}{\mu_{IO,j} - \sum_{i=1}^{n} \left(x_{ij} \cdot \lambda_{IO,i}^{\text{primary}} + y_{ij} \cdot \lambda_{IO,i}^{\text{standby}} \right)}$$

ここで:

- R: 全ホストの総レスポンスタイム
- m: 総ホスト数
- n: 総サービス数
- x_{ij}: サービス i の現用系 VM がホスト j に割り当てられている場合 1. それ以外は 0 となるような関数.

- y_{ij}: サービス i の予備系 VM がホスト j に割り当てられている場合 1. それ以外は 0 となるような関数.
- $\lambda_{CPU,i}^{\text{primary}}$: VM i の現用系 CPU 到着率
- $\lambda_{IO,i}^{\text{primary}}$: VM i の現用系 I/O 到着率
- $\lambda_{CPU,i}^{\text{standby}}$: VM i の予備系 CPU 到着率
- $\lambda_{IO,i}^{\text{standby}}$: VM i の予備系 CPU 到着率
- $\mu_{CPU,j}$: ホスト j の CPU サービスレート
- $\mu_{IO,j}$: ホスト j の I/O サービスレート
- a, b: 影響度

4.2　制約条件

4.2.1　制約条件 1

到着率がサービス率を超えないこと:

$$\lambda_{CPU,j} \leq \mu_{CPU,j}, \quad \lambda_{IO,j} \leq \mu_{IO,j} \quad \forall j$$

ここで

- $\lambda_{CPU,j} = \sum_{i=1}^{n} \left(x_{ij} \cdot \lambda_{CPU,i}^{\text{primary}} + y_{ij} \cdot \lambda_{CPU,i}^{\text{standby}} \right)$
- $\lambda_{IO,j} = \sum_{i=1}^{n} \left(x_{ij} \cdot \lambda_{IO,i}^{\text{primary}} + y_{ij} \cdot \lambda_{IO,i}^{\text{standby}} \right)$

制約条件 1 つ目は到着率がサービス率を超えないことである. これは 1 秒間のサーバーへのリクエスト数が, 1 秒間にサーバーが処理できるリクエスト数を超えている状態を表す. これは待ち時間が増え続けるような状態となり待ち行列理論において不適である. そのためそのようなホストが存在しないように制約条件を設ける.

4.2.2　制約条件 2

現用系 VM と予備系 VM が同じホストに配置されないこと:

$$x_{ij} + y_{ij} \leq 1 \quad \forall i, \forall j$$

制約条件 2 は，サービスの現用系と予備系を同じ配置にしないことである [4]．この制約条件は，デュプレックスシステムにおいて，一般的な条件である．具体例を図 4.1 に示す．図は，サービス A，B を配置した様子を表している．この配置では，ホスト 1 で障害が発生した場合，サービス A は ホスト 2 にある予備系に処理が移るが，サービス B は予備系も同時に障害に合うため，可用性が向上しているとはいいがたい．そのため，サービス A のように，現用系と予備系を同じホストに配置しない．

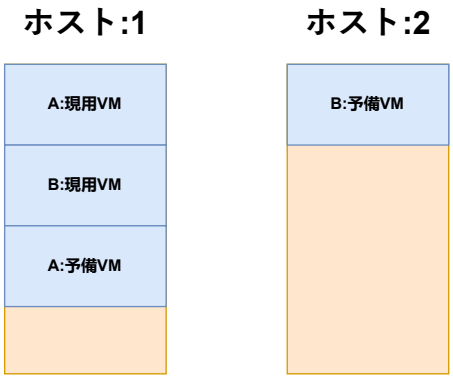

図 4.1　制約条件 2 の具体例

4.2.3　制約条件 3

他の VM と配置パターンが重複しないこと:

$$((x_{kj} + x_{lj} \leq 1) \vee (y_{kj} + y_{lj} \leq 1)) \wedge ((x_{kj} + y_{lj} \leq 1) \vee (y_{kj} + x_{lj} \leq 1)) \quad \forall k, l \quad (k \neq l)$$

制約条件 3 は，違うサービスの現用系と予備系の配置と同じにしないことである [4]．この制約条件は，可用性を考慮した条件である．具体例を図 4.2 に示す．デュプレックスシステムでは，障害が発生した際に，予備系に処理が移動することで可用性を高くしている．図のサービス A，B のような状態の際に，ホスト 1 で障害が発生した場合，サービス A と B に

処理が移動するため，ホスト 2 のリソース要求が高くなることが予測される．さらには，ホスト 2 まで処理落ちしてしまう危険性もある．そのため，サービス A，B のように，現用系と予備系の配置の組み合わせが重複するような配置は行わないように制約条件を設ける．

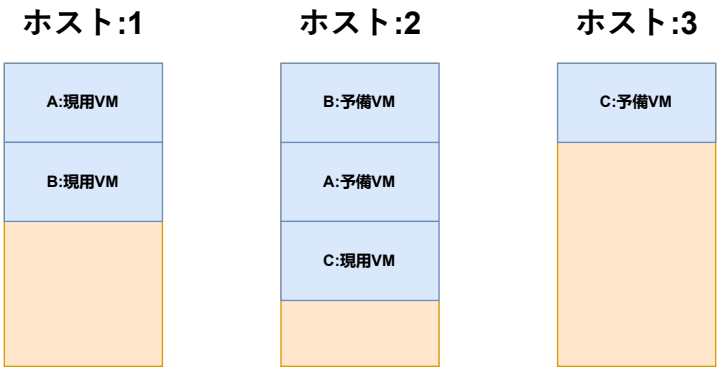

図 4.2　制約条件 3 の具体例

4.3　提案手法の基本的な考え方

VM を再配置するにあたって, レスポンスタイムへ与える上昇量が同程度となるような配置にすることによってレスポンスタイムに対して最も効率が良くなるような配置を実現できる. なぜなら一例として, 上昇量が I/O の方が多い状態を考えると, 新しく I/O の到着率の高い VM が配置された場合, I/O が大きなボトルネックとしてレスポンスタイムの劣化につながる. そのため上昇量が同程度の到着率とすることによってレスポンスタイムへ与える影響を抑えながら影響度の低い到着率を上昇させることができる.

具体的に, I/O のレスポンスタイムへ与える影響が CPU の 2 倍である場合を考える. この時それぞれの到着率がレスポンスタイムに与える上昇量はそれぞれ以下のレスポンスタイムの式に $a = 1, b = 2$ を代入し, それぞれの到着率 (CPU:$\lambda_{CPU}, I/O : \lambda_{I/O}$) で偏微分した値となる.

$$ResponseTime = \frac{a}{\mu_{CPU} - \lambda_{CPU}} + \frac{b}{\mu_{I/O} - \lambda_{I/O}}$$

よってその上昇量は以下のように計算でき, また, その概形は 4.3 のようになる.

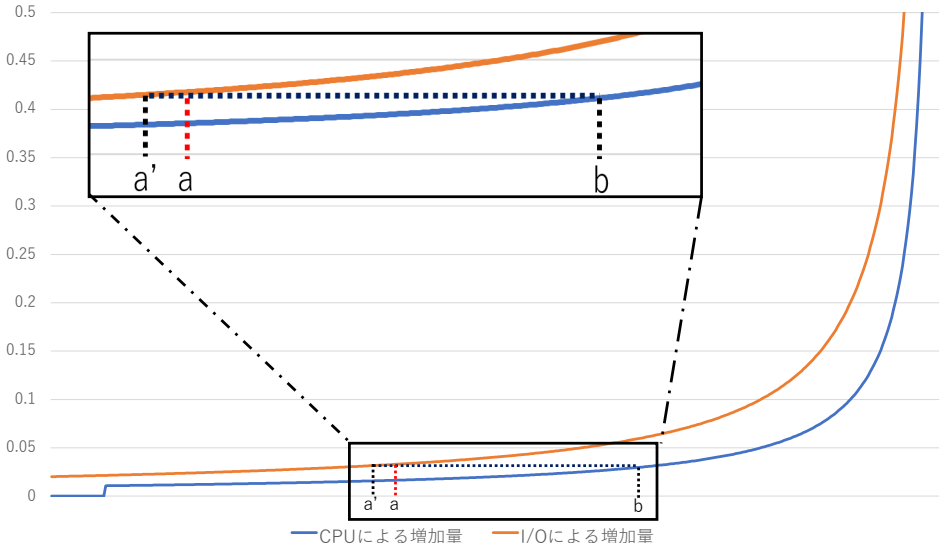

図 4.3　上昇量のグラフの概形

CPU の上昇量:

$$\frac{\partial}{\partial \lambda_{CPU}}\left(\frac{1}{\mu_{CPU}-\lambda_{CPU}}+\frac{2}{\mu_{I/O}-\lambda_{I/O}}\right)$$
$$=\frac{1}{(\mu_{CPU}-\lambda_{CPU})^2}$$

I/O の上昇量:

$$\frac{\partial}{\partial \lambda_{I/O}}\left(\frac{1}{\mu_{CPU}-\lambda_{CPU}}+\frac{2}{\mu_{I/O}-\lambda_{I/O}}\right)$$
$$=\frac{2}{(\mu_{I/O}-\lambda_{I/O})^2}$$

この時,I/O の到着率を a ’ とした場合 I/O の上昇量が CPU の上昇量が同程度となるのは到着率が b の時である. ここで, I/O 到着率を a ’ , CPU 到着率を b とした配置にすることによって多くの到着率を占めながらレスポンスタイムに対して効率的に配置することができる.

その一方, 上昇量が同程度になるような配置ができなかった VM については I/O の到着率が平均と比べ少し上昇するが, CPU の到着率を平均と比較して著しく低下させることができる. そのような VM については, できるだけ CPU と I/O が平滑になるように配置する

ことによって CPU, I/O の偏りを防ぐことができ, レスポンスタイムへの影響を少なくできる. そのため, これらの VM については CPU と I/O の比率の逆順でペアリングを行うことによって平滑的な配置を行うことによってレスポンスタイムの低下を目指す.

このようなレスポンスタイムの上昇率が同程度になるような配置を行うが負荷率が高くなるようなグループと, 上昇量に対する効率は悪いが負荷率が低くなるような 2 つのグループに分けることによってレスポンスタイムの低下を目指す.

4.4　再配置の手法

以下に提案手法の処理の内容について説明し, 具体的に疑似コードを用いて提案手法の概略を示す.

大きく以下の流れに基づいて再配置を行う.

1. 現在の VM の配置をすべて解除する
2. 上昇量が同程度となるような平均到着率に対する比率 (α, β) を求める
3. レスポンスタイムの上昇量が同程度となるようなホスト (疑似コード内:Group1) の配置を行う
4. 平滑的な配置を行うホスト (疑似コード内:Group2) の配置を行う

疑似コードは以下のとおりである.

Algorithm 1 Optimize VM Placement with Service Constraints

1: **procedure** OPTIMIZEPLACEMENT
2: Clear all existing VM allocations from hosts
3: Calculate optimal α and β using CALCULATEOPTIMALALPHABETA
4: Divide hosts into two groups: *Group*1 (Best-Fit) and *Group*2 (Balanced-Load)
5: Place VMs in *Group*1 using PLACEBESTFIT with α, β, and service constraints
6: Place remaining VMs in *Group*2 using PLACEBALANCEDLOAD with service constraints
7: **end procedure**

4.4.1 上昇量が同程度となるような平均到着率に対する比率の計算

4.3 で述べたような上昇量が同程度となる到着率の計算を行う.

ここではレスポンスタイムへの影響度の大きいもの (CPU または I/O) を平均到着率に対して α 倍, 小さいものを β 倍したものが等しい点を微小範囲で精査する. もし求めた $\alpha + \beta$ が現状の最大値よりも大きければ更新を行い, 最大値を見つける. 負荷率が高いような状況では $\alpha + \beta \leq 2$ といった平滑的な配置を行った場合に比べて到着率が少ないような場合が生じうる. このような場合はレスポンスタイムへの勾配を用いた配置を行うことによる効率化は期待できない. そのため $\alpha = \beta = 1$ としてホストのレスポンスタイムの上昇量が同程度となるような到着率を目指すグループ, その他のグループといったグループ分けを行わず, 全体を平滑的に配置する.

疑似コードは以下のとおりである.

Algorithm 2 CalculateOptimalAlphaBeta

```
1: procedure CalculateOptimalAlphaBeta
2:     Compute average CPU and I/O arrival rates from all VMs
3:     Set μ_CPU and μ_IO as service rates
4:     Initialize α and β as 0
5:     for each combination of α and β in valid ranges do
6:         Compute the partial derivatives of response times
7:         if Partial derivatives satisfy balancing condition and constraints then
8:             Update α and β to maximize their sum
9:         end if
10:    end for
11:    if No valid α and β found then
12:        set defaults α = 1, β = 1
13:    end if
14:    return α and β
15: end procedure
```

4.4.2 レスポンスタイムの上昇量が同程度となるようなホストの配置

次に先ほど求めた α, β を基にベストフィットによる配置を行う.

まず, すでに配置された VM でないかチェックを行う. 次に制約条件として設けている到着率がサービス率を超えないように配置後の到着率がサービス率を超過することがないかの確認を行う. 同様に予備系ホストについて可用性の制約である制約条件 2, 3 の確認を行う.

制約条件を満たすようなホストを絞り込み, レスポンスタイムの上昇量が同程度となるような到着率を用いたベストフィットディクリーシングアルゴリズム (以降 BFD と略す) によって配置を行う. 具体的には VM を総到着率 ($\lambda_{CPU} + \lambda_{I/O}$) の降順にソートし, 上昇量が同程度になるような到着率に配置するグループに属するホストも同様に負荷率

の降順にソートを行う.VM を順次ホストに仮配置を行っていき, 計算によって求められた上昇量が同程度となるような到着率を IdialCPU(I/O)ArriveRate, 仮配置した到着率を TempolaryCPU(I/O)ArriveRate として以下の値を計算する.

$$|IdialCPUArriveRate - TempolaryCPUArriveRate|$$

$$+|IdialI/OArriveRate - TempolaryI/OArriveRate|$$

この値が最も小さいホストを選択し配置処理を行う. そして, 配置された VM を配置済み VM として登録する. 以下同様に任意の上昇量が同程度となるような到着率に配置するグループに属するホストの実際の到着率と計算によって求めた到着率との差が閾値未満になるまで配置を行う.

このように BFD アルゴリズムを用いることによって上昇量が同程度となるような配置に近づけることができる.

疑似コードは以下のとおりである.

Algorithm 3 PlaceBestFit with Service Constraints

procedure PLACEBESTFIT(*Group*, α, β)
  Compute target CPU and I/O loads using α and β
  for each VM in the system **do**
    if VM is already placed, skip it **then**
    else
      Identify the best-fit host in *Group* for the VM based on load difference
      Check general constraints:
        $Host.cpuLoad + VM.cpuArrivalRate \leq \alpha \cdot AvgCpuLoad + \text{threshold}$
        $Host.ioLoad + VM.ioArrivalRate \leq \beta \cdot AvgIoLoad + \text{threshold}$
      Check service-specific constraints:
        Current and standby VMs of the same service are not on the same host
        Current and standby VMs of other services do not follow the same placement pattern
      if A valid host is found and all constraints are satisfied **then**
        Allocate VM to the host and mark it as placed
      end if
    end if
  end for
end procedure

4.4.3 平滑的な配置を行うホストの配置

その他の VM については平滑的に配置することによってレスポンスタイムの低下を目指す.

平滑的に配置するにあたって以下の図 4.4 のように CPU 到着率割合の降順と I/O 到着率

割合の降順にソートしたリストを用意する.2 つの VM のリストの最初の VM を取得しペアリングを行うことによって負荷の平滑化が目指せる. そのため, 図内のように現用系, 予備系ともにペア VM を形成する.

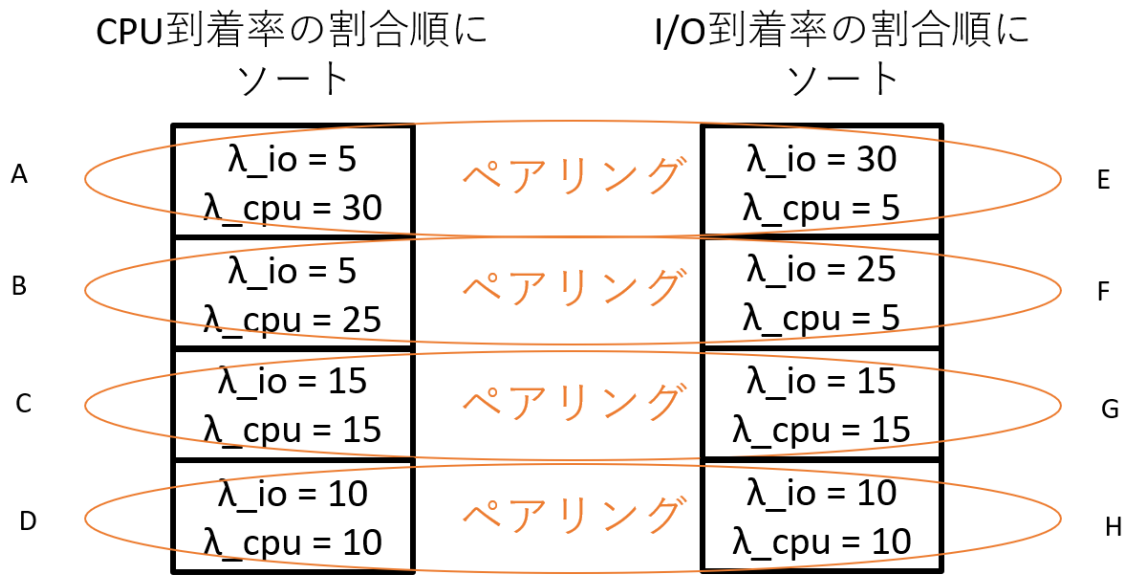

図 4.4　VM のペアリング

このペア VM を負荷率の総和の降順でソートする. 同様にホストもソートし, 制約条件の確認を行う. 制約条件を満たすような最初のホストへの配置を行う.

ここで, 予備系の VM と現用系の VM が同じペアを形成する場合について考える. すると以下の図 4.5 のように, どの VM に配置しても制約条件 3 である「他の VM と配置パターンが重複しないこと」を満たすことができなくなる. そのためペア VM の制約として予備系の VM は現用系の VM と同じペアを形成しないように確認を行う.

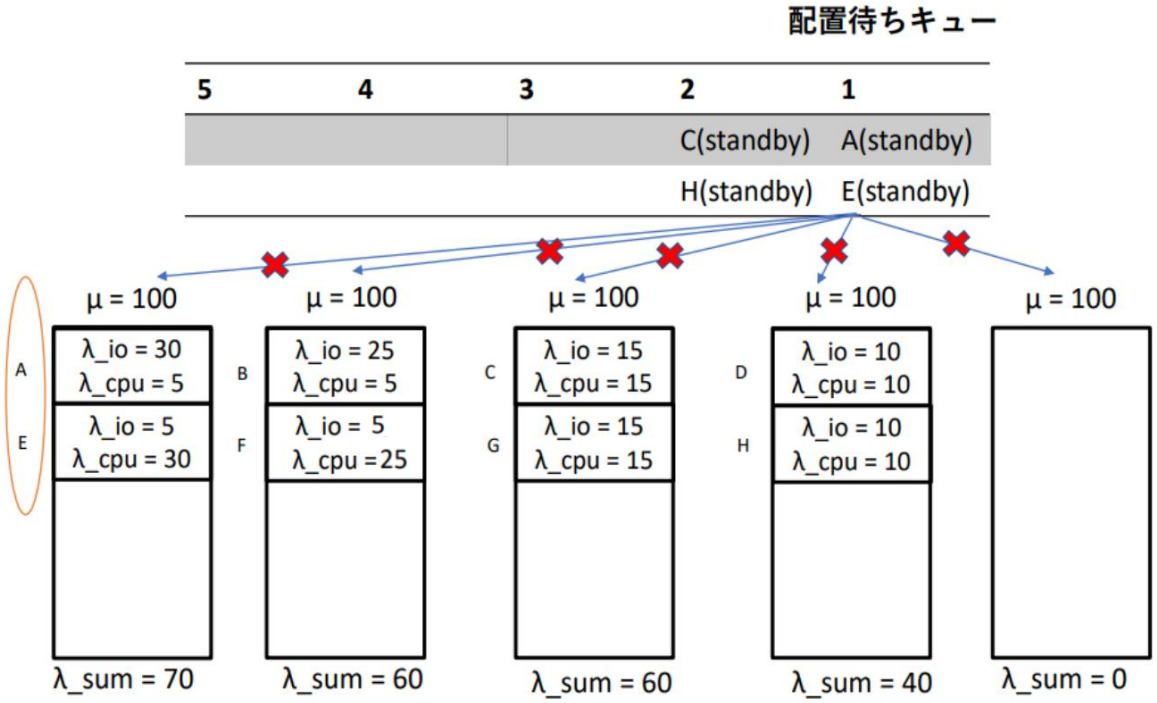

図 4.5　ペア VM の制約

疑似コードは以下の通りである.

Algorithm 4 PlaceBalancedLoad with Pair Constraints

1: **procedure** PLACEBALANCEDLOAD(*Group*)
2: Sort *Group* hosts by their load
3: Sort VMs by CPU-to-I/O and I/O-to-CPU ratios
4: **for** each unplaced VM pair (high CPU and high I/O ratio VMs) **do**
5: Identify the host with the lowest total load in *Group*
6: Check general constraints:
7: $Host.cpuLoad + VM_{CPU}.cpuArrivalRate + VM_{IO}.cpuArrivalRate \leq$ CPU capacity
8: $Host.ioLoad + VM_{CPU}.ioArrivalRate + VM_{IO}.ioArrivalRate \leq$ I/O capacity
9: Check service-specific constraints:
10: Current and standby VMs of the same service are not on the same host
11: Current and standby VMs of other services do not follow the same placement pattern
12: Check pair constraints:
13: Ensure VM_{CPU} and VM_{IO} are not paired in both current and standby configurations
14: **if** The host can accommodate the VM pair and all constraints are satisfied **then**
15: Allocate the VM pair to the host
16: **end if**
17: **end for**
18: **end procedure**

第 5 章

シミュレーション環境

ここでは提案手法を評価するシミュレーション環境について説明する.

5.1 環境想定

ホスト数に最大値があるような環境で, ランダムな時間で一定確率でサービスが発生するようなシミュレーションをおこなう.VM の初期配置は VM の到着率の降順にファーストフィットディクリーシング (FFD) アルゴリズムを用いて配置するものとする. またホストは同機種クラスタを想定しているため, サービス率はすべて等しく, VM の到着率はランダムな値で既知であるとする. 予備系の到着率はホットスタンバイを想定しており, 現用系と同期処理をする必要性がある. そのため, 現用系対して一定割合の到着率を持つものとする.

5.2 パラメータ

シミュレーションを行うにあたってのパラメータについて説明する. シミュレーション全体のパラメータは以下のとおりである.

表 5.1　全体パラメータ

情報	値
サービス発生間隔	20 分
サービス発生確率	15%
最大サービス発生数	20
シミュレーション時間	10 日
再配置間隔	1 日
障害が発生する確率	1/10, 1/100, 1/1000(1/day)

5.2.1　VM とホストに関するパラメータ

VM とホストが持つ各パラメータについて以下の表に示す.

表 5.2　ホストに関するパラメータ

情報	値
ホスト数	100
CPU サービス率	1500
I/O サービス率	1500

表 5.3　VM に関するパラメータ

情報	値
CPU 到着率	0 - 350
I/O 到着率	0 - 350
予備系の到着率割合	0.2

5.2.2　復旧時間に関するパラメータ

障害を復旧するにあたって, 目標復旧時間 (PRO) はシステムによって異なる.　また同様に, 予備系の同期情報から復旧するまでに要する時間も異なる. ここでは復旧するまでの時間は以下の表のようなパラメータで実装する.

表 5.4　復旧時間

条件	最小復旧時間	最大復旧時間
予備系が障害にあっていない	30s	5m
予備系も障害にあっている	2h	2.5h

第 6 章

評価

提案する配置手法の有効性を確認するため，シミュレーションにより評価を行う．以下，評価方法と結果を示す.

6.1　評価方法

提案する配置手法および，比較手法の配置アルゴリズムを実装し，シミュレーションを行う．シミュレーターでは，物理マシンと仮想マシンを用意し，物理マシンのリソース容量を超えないように仮想マシンを配置し，疑似的に一定時間運用する．このシミュレーションより，負荷率を監視し, クラウドコンピューティング環境の負荷率に対するレスポンスタイムや可用性を検証する.

また, 到着率の割合が大きく異なると, ホストの到着率の偏りが生じ, どの手法においてもレスポンスタイムの悪化が予想される. そのため, 本手法がどの程度の偏りまで有効であるか到着率の割合を変更し, 検証をおこなう.

さらに，シミュレーション中にいくつかのの物理マシンに障害を発生させ，その際の復旧時間を評価し, 可用性評価についても検討する.

6.1.1　評価指標

評価指標には, 以下の指標を用いる.

1. レスポンスタイム (負荷率特性)
2. 障害復旧時間 (合計および平均)

3. 稼働率

ここで負荷率はすべてのホストの CPU と I/O を合計した負荷率の平均を示し, 以下のように計算する.

$$\mathrm{AverageLoadFactor} = \frac{1}{2m}\sum_{j=1}^{m}\frac{\lambda_{CPU,j}}{\mu_{CPU,j}} + \frac{\lambda_{IO,j}}{\mu_{IO,j}}$$

レスポンスタイムはすべてのホストの平均レスポンスタイムを評価する. 具体的には以下の式によって算出した値について評価を行う.

$$\mathrm{AvaregeResponseTime} = \frac{1}{m}\sum_{j=1}^{m}\frac{a}{\mu_{CPU,j} - \sum_{i=1}^{n}\left(x_{ij}\cdot\lambda_{CPU,i}^{\mathrm{primary}} + y_{ij}\cdot\lambda_{CPU,i}^{\mathrm{standby}}\right)} + \frac{b}{\mu_{IO,j} - \sum_{i=1}^{n}\left(x_{ij}\cdot\lambda_{IO,i}^{\mathrm{primary}} + y_{ij}\cdot\lambda_{IO,i}^{\mathrm{standby}}\right)}$$

次に復旧時間についてはあるホストが故障した場合そのホストに配置されている VM がすべて復旧するまでの時間について評価した. 復旧する時間は現用系が障害が発生した場合, 予備系が障害が起きている場合は現用系が復旧するまでの時間とし, 予備系は障害が起きていない場合は同期情報から復旧する時間を評価する. また稼働率は, すべてのホストのシミュレーション上の時間の総和 (総時間), 任意のホストが障害が発生してから復旧するまでに要した時間の総和 (総障害復旧時間) を用いて以下の式で評価する.

$$\text{稼働率} = \frac{\text{総時間} - \text{総障害復旧時間}}{\text{総時間}}$$

6.1.2　比較手法

提案手法の有用性を評価するために以下を比較対象として以下を用いる

1. 提案手法
2. 再配置しない場合
3. グリーディ法
4. 遺伝的アルゴリズム (制約条件なし)

ここで遺伝的アルゴリズムは世代数が十分大きい場合最適解に十分近似することが知られている. そのため, ここでは, 制約条件をなくし, 十分世代の進んだ遺伝的アルゴリズムを用いることによって疑似的なレスポンスタイムの最適解として評価を行う. 遺伝的アルゴリズムの遺伝子選択手法をトーナメント式として, パラメータは以下の値を用いる.

表 6.1　遺伝的アルゴリズムのパラメータ

情報	値
保存する遺伝子数	10000
世代数	500000
交叉率	0.8
突然変異率	0.9 - 0.1

6.2　結果と考察

6.2.1　CPU と I/O の到着率が同程度の場合の結果と考察

影響度が同程度の場合を図 6.1 に,　異なる場合を図 6.2 に示す. 図 6.1 においても, 図 6.2 においてもグリーディ法による再配置や再配置をしない場合と比べてレスポンスタイムの低下を確認することができた. また, 制約条件をなくした遺伝的アルゴリズムと比較しても, 影響度が同程度の場合は負荷率が 70%までは同等のレスポンスタイムで, 80%に近いところでも 1.3 倍程度に抑えることができている. 影響度が異なる場合についても負荷率が 65%程度までは同等のレスポンスタイムで, 80%に近いところでも 1.4 倍程度に抑えることができているとわかる.

6.2　結果と考察

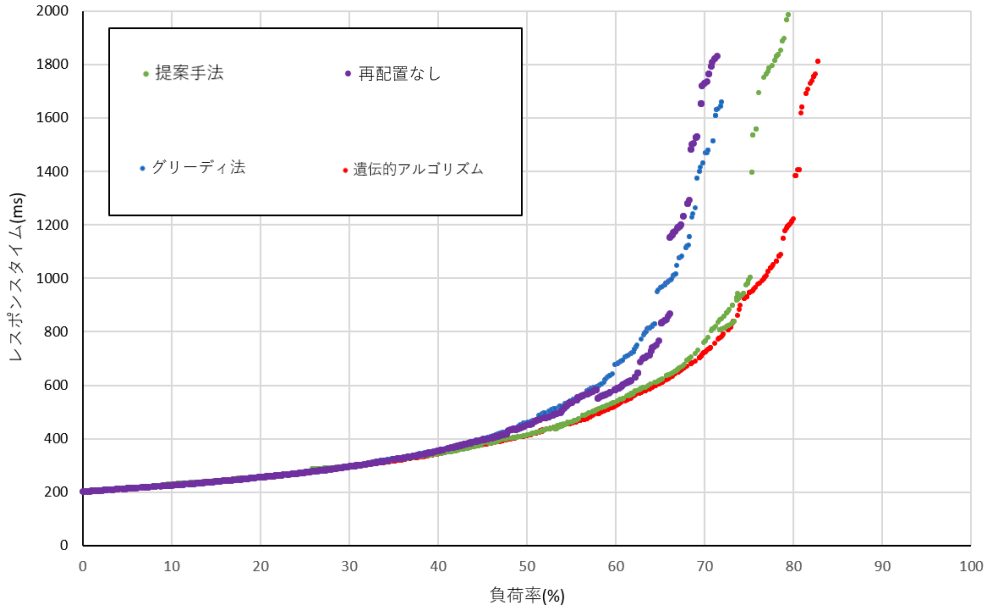

図 6.1　影響度 1:1 の負荷率特性

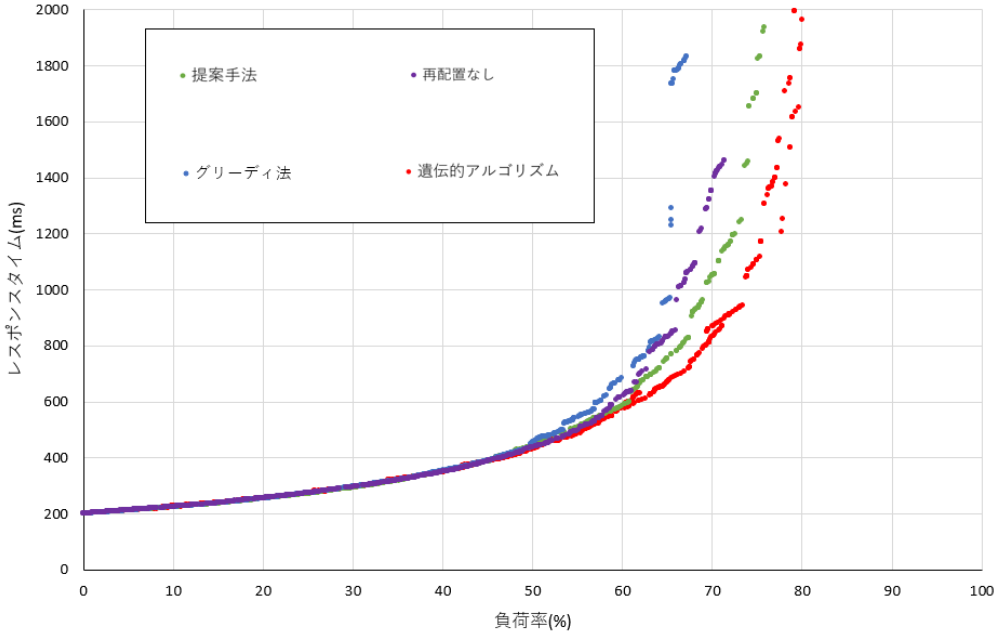

図 6.2　影響度 1:2 の負荷率特性

6.2.2　到着率が異なる場合の結果と考察

以下は I/O の到着率が 1.25 倍の場合の結果を図 6.3, 1.5 倍程度であるときの結果を図 6.4 に示す.1.25 倍の時は同程度の時と比べて効果は少ないが, 本再配置手法による恩恵がある. しかし 1.5 等大きく到着率が異なると, 再配置することによる効果は確認できるが, 手法による大きな差は生じなかった. これは偏りによって配置できる VM の数が減少し, 再配置を行う手法においては再配置の際にソートし配置しなおすによってある程度効率的に VM を配置することはできるが, ホスト内の到着率の偏りを抑えることは難しいためであると考えられる. また, レスポンスタイムについても到着率と同程度の時と比較して増加していることがわかる. これも先ほどと同様にホスト内の到着率の偏りにを抑えることは難しく, 比較的到着率の高い傾向にある I/O の高負荷によってレスポンスタイムが大きく悪化したためであると考えられる.

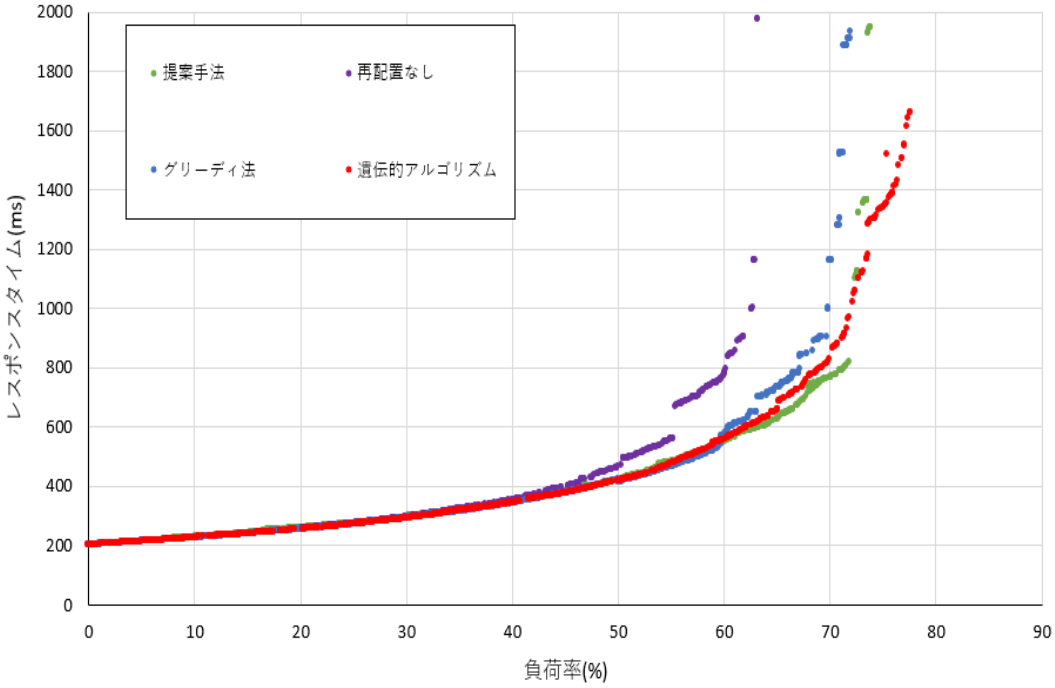

図 6.3　I/O 到着率:1.25 倍

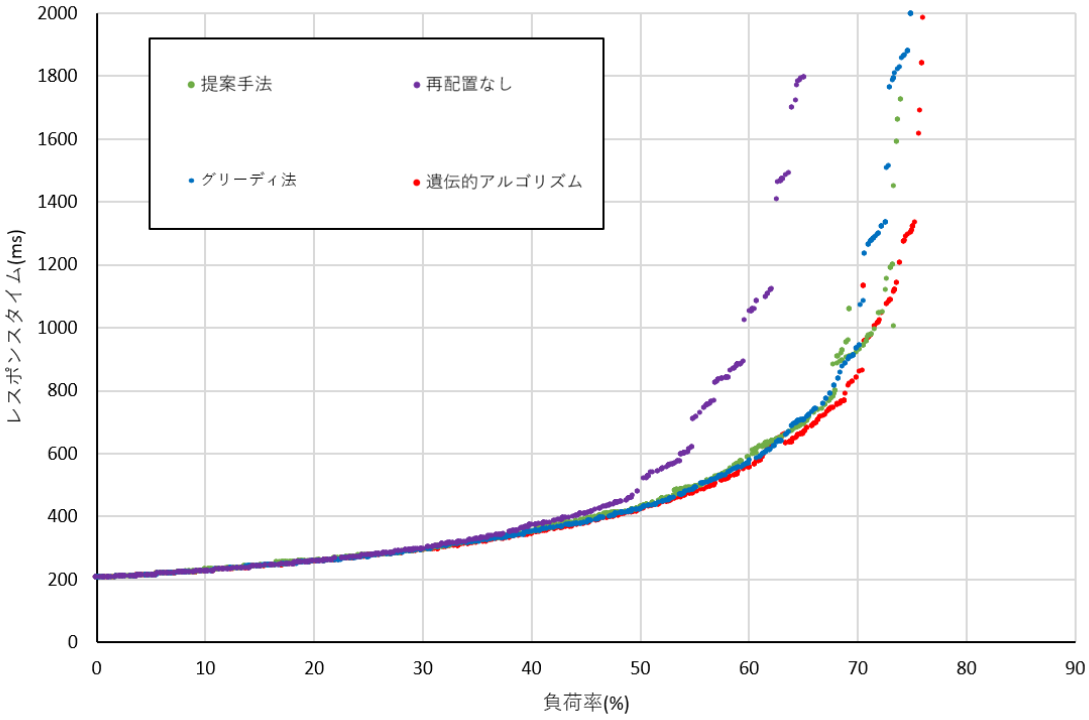

図 6.4　I/O 到着率:1.5 倍

6.2.3　可用性に関する結果と考察

総障害時間の結果を図 6.5 に, 平均復旧時間の結果を図 6.6 に, 稼働率の結果を表 6.2 に示す. 本手法実現によって障害が発生する確率が 1/10(1/day), 1/100(1/day) 条件において障害復旧時間と稼働率の向上が確認できた. 特に障害が発生する確率が 1/100(1/day) の条件においては 50%程度の障害復旧時間の向上が確認できた. また同様の条件において図 6.6 を確認してみると, 制約条件なしの条件においては平均復旧時間が上昇している一方, 制約条件ありにおいては障害が発生する確率 1/1000(1/day) の条件と同等の平均復旧時間となっている. このことから制約条件の設定によって障害の発生回数は増加によって総復旧時間は増加しているものの, 平均復旧時間は 1 日当たりの障害が発生する確率 1/1000 と十分低い条件と同程度に抑えることができていることがわかる. 障害が発生する確率 1/10(1/day) の条件では 1/100(1/day) の条件と比較すると大きな差は生じなかった. これは障害が発生する確率 1/10(1/day) 条件では障害が発生する確率が高く, 制約条件により別のサーバーへ予

6.2　結果と考察

備系を設置していても予備系も障害によって止まっているような状態が発生しやすくなっているからであると考えられる. 障害が発生する確率 1/1000(1/day) においては障害復旧時間の減少は確認できるが, 大きな差はみられなかった. これは障害が発生する確率がホスト数に対して低く, 制約条件によって別の組み合わせで配置しない状態でも予備系が生存する確率が高く, 制約条件が大きな意味をなしていなかったからであると考えられる.

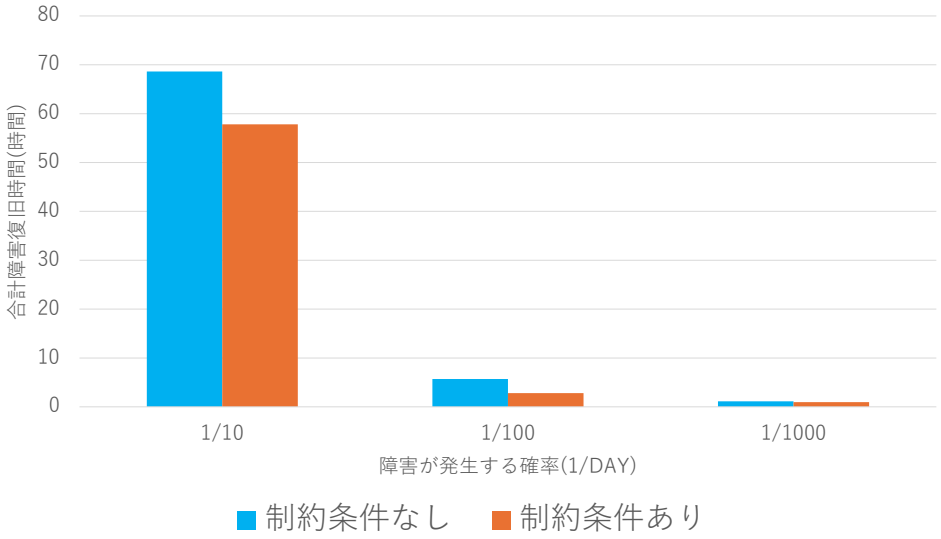

図 6.5　合計障害復旧時間

6.2　結果と考察

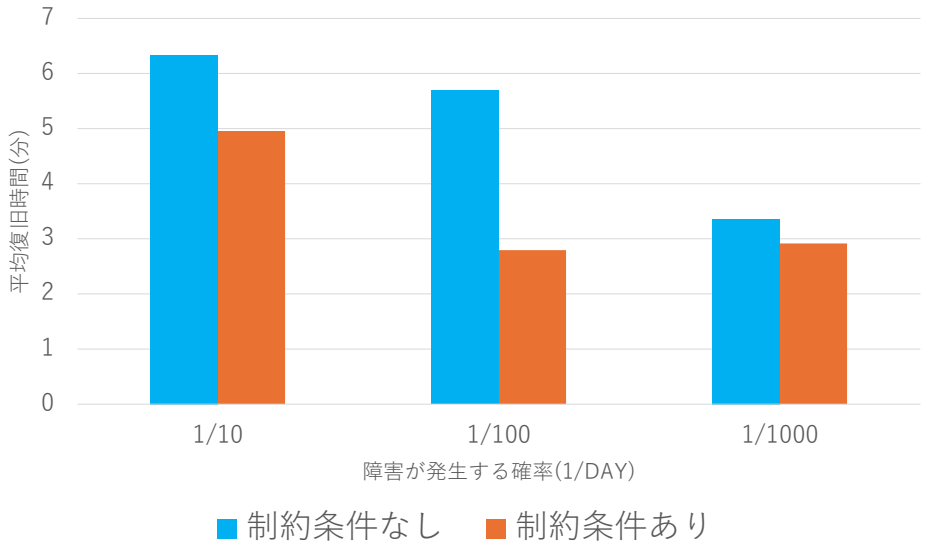

図 6.6　平均障害復旧時間

表 6.2　稼働率

(障害が発生する確率, 1/day)	稼働率 (制約条件なし, %)	稼働率 (制約条件あり, %)
1/10	99.982658	99.971460
1/100	99.997628	99.998836
1/1000	99.999533	99.999595

第 7 章

おわりに

7.1 まとめ

近年, クラウドコンピューティングが急速に普及している. クラウド事業者は安定したサービスを提供するにあったって SLA として稼働率を保証する場合が多い. また近年, クラウドアプリケーション提供者は SLO というサービスが満たすべき目標を設定する場合が多く, SLO においてレスポンスタイムが設定される場合が多い. このように可用性を満たしながらレスポンスタイムを最小化するような手法が求められている.

菊森らは、可用性を考慮しながら電力の最小化を目指す動的配置手法について検討している. この研究において, ホストの高負荷による電力上昇について考えられている. しかし, 高負荷によるレスポンスタイムの上昇については考慮されておらず, レスポンスタイムについての評価も行われていない.

そこで, 本研究では可用性を考慮しながら, レスポンスタイムの最小化を目指す再配置手法について提案し, 有用性を評価を行った. 評価では, 提案手法, 再配置なし条件, グリーディ法, 制約条件のない遺伝的アルゴリズムを用いてレスポンスタイムと障害復旧時間, 稼働率の比較を行った. その結果提案手法の有効性を示した.

7.2 今後の課題

本研究において VM の CPU, I/O の到着率が既知であるとしている. そのため, 事前に VM の特性が調査済みであるような環境が想定されており現実的とは言い難い.

また, 本研究において到着時間をランダムにシミュレーション上で生成している. そのため, 非現実的なパラメータも生じうる. よって, より現実に近いシミュレーションを行うために, 実際のデータセンターでのデータセットなどに即してシミュレーションすべきである.

さらに, 提案手法では到着率が大きく異なる場合について, レスポンスタイム向上は図れなかった. そのため I/O バウンドのサービスが極端に多いなど, 特定の環境におけるアルゴリズム等は別途検討する必要がある.

謝辞

本研究を進めるにあたり，修士研究の指導教員としてご指導いただきました横山和俊教授には丁寧にご指導を頂き，大変お世話になりました．心より感謝いたします．本研究の副査をお引き受けいただきました，松崎公紀教授，高田喜朗教授に感謝いたします．同期，後輩の方々も研究室活動などの様々な面でお世話になりました．ありがとうございました．

参考文献

[1] 総務省, ”令和 6 年度版情報白書”, < https://www.soumu.go.jp/johotsusintokei /whitepaper/ja/r06/html/nd218200.html >, (2025 年 1 月閲覧).

[2] ”Amazon Web Services, Inc.”, ”Amazon Compute SLA”, ¡https://d1.awsstatic.com/legal/AmazonComputeServiceLevelAgreement /Amazon_Compute_Service_Level_Agreement_English_2022-05-25.pdf¿,(2025 年 1 月閲覧).

[3] 株式会社メルカリ, ”メルペイにおける SLO の活用事例 – 信頼性を定義しよう”, ¡https://engineering.mercari.com/blog/entry/20210928-mtf2021-day4-3/¿,(2025 年 1 月閲覧)

[4] 菊森剛, 横山和俊, クラウドコンピューティングにおける可用性と消費電力を考慮した仮想マシン配置手法の提案, 情報処理学会 86 回全国大会 1X-02, 2024).

[5] Mahdi Mollamotalebi and Shahnaz Hajireza. Multi-objective dynamic management of virtual machines in cloud environments. Journal of Cloud Computing, Vol. 6, No. 1, p. 16, Jul 2017.

[6] Weizhe Zhang, Hongli Zhang, Huixiang Chen, Qizhen Zhang, Improving the QoS of Web Applications across Multiple Virtual Machines in Cloud, 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012.

[7] Cheng Li, Nitro: A capacity-optimized SSD cache for primary storage,2014 USENIX Annual Technical Conference 978-1-931971-10-2(2014).

[8] Ji Xue, Feng Yan, Alma Riska,Evgenia Smirni,Scheduling data analytics work with performance guarantees: queuing and machine learning models in synergy, Springer Science+Business Media New York, Volume 19, pages 849–864, (2016).

[9] N. Thirupathi Rao, PillaSrinivas, K. Sudha, Debnath Bhattacharyya, ai-Hoon Kim : Performance of M/M/1 and M/D/1 Queuing Models on Data Centers with Cloud Computing Technology Using MATLAB, International Journal of Grid and Distributed Computing Vol. 11, No. 3 (2018), pp.11-22.

[10] Khaled M. ELLEITHY, Using a Queuing Model to Analyze the Performance of Web Servers, Department of Computer Science and Engineering University of Bridgeport Bridgeport, CT 06611(2009).