Vol.2020-OS-148 No.9
2020/2/28

通信経路を考慮したマイクロサービスの高速化検討

横山 遼[1]　坂本 龍一[2]　中村 宏[2]

概要：複数の物理マシンにまたがって動作するマイクロサービスによるアプリケーションで，通信オーバーヘッドが問題となっている．物理マシン間通信削減のためマイクロサービスを少数マシンに集約配置すると，リソース使用量に偏りが生じ一部のマシンがボトルネックとなりスループットが低下しうる．本研究ではリソース使用量の複数マシンへの配分と物理マシン間通信の削減をアプリケーションに合わせたバランスで行う，マイクロサービスの物理マシンへの最適配置アルゴリズムを提案する．その後，既存のマイクロサービスアプリケーションにも容易に導入可能な最適化システムおよび評価環境を実装した．ベンチマークを用いた性能評価では，マイクロサービス動作環境のデファクトスタンダードである Kubernetes スケジューラと比較してより均一なリソース使用量配分および通信時間削減を果たし，スループット・レイテンシ共に改善を果たした．

1. はじめに

近年マイクロサービスアーキテクチャの利用が急速に進んでいる [1]．マイクロサービスアーキテクチャではアプリケーションを小さなサービス=マイクロサービスに分割し，それぞれを異なるプロセスとして動作させる．

従来の開発手法であるモノリシックでは，全ての機能がまとめて 1 つのプロセスとして同一マシン上で実行される．一方マイクロサービスアーキテクチャでは各マイクロサービスは独立したコンテナとして別々に存在し，それらが連携して一つのアプリケーションをなす．

マイクロサービスアーキテクチャでは多数のコンテナにより莫大なマシンリソースが消費されるため，マイクロサービスは複数のマシンに分散して存在し，物理マシン間での通信オーバーヘッドが問題となっている [2]．リアルタイム性の要求されるアプリケーション，例えば金融取引や自動運転等の分野では ms 単位のレイテンシが問題となるため，マイクロサービスアーキテクチャの応用は進んでいない．

また，現在マイクロサービスが多く取り入れられている Web サービス分野などではスループットの向上により高い負荷耐性を担保することも重要である．

そこで本研究では，通信を考慮したマイクロサービスの物理マシンへの最適配置によるアプリケーションのスループットの向上，各リクエストに対するレスポンスのレイテンシ削減を目指す．

1　東京大学工学部計数工学科
2　東京大学大学院情報理工学系研究科

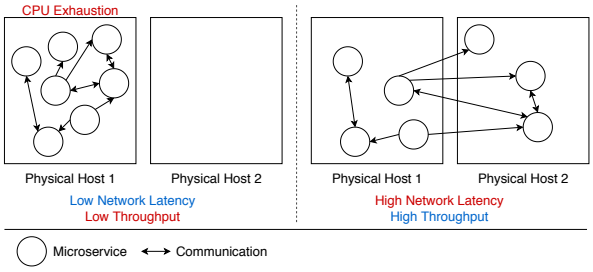


図 1　リソース使用と通信のトレードオフ

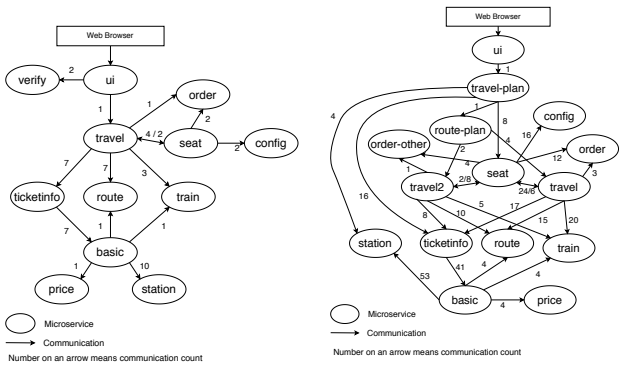
2. マイクロサービスの最適配置問題

最適配置において，リソース使用率の分配および物理マシン間通信の削減は重要なファクターとなる．CPU やディスク I/O 等のリソース使用率に物理マシン間で偏りが生じると，負荷の大きくなったマシンでリソースが枯渇しスループットが致命的に悪化しうる．また，物理マシンをまたぐ通信はレイテンシ増大につながる．

この 2 要素はトレードオフの関係にある．例えば少数のマシンにマイクロサービスを集約すると物理マシン間の通信は削減されるが，特定のマシンでリソースの枯渇を引き起こす確率が高くなる．一方で多数のマシンに分散配置するとマシン間通信のオーバーヘッドが増大する (図 1)．

実環境におけるスループットとレイテンシの優先度付けはアプリケーションに依る．アプリケーションの特性や目的に合わせた配置最適化を行うことのできる汎用的なシステムの構築は喫緊の課題である．

マイクロサービスアーキテクチャではアプリケーションやリクエスト状況によって各サービス対ごとに生じる通信



(a) BasicSearch (b) AdvancedSearch

図 2 リクエストによる通信回数，稼働マイクロサービスの違い図示していないが，通信容量もリクエストごと，サービス対ごとに大きく異なる

回数・容量，各サービスのリソース使用量が動的に変化する．そのため，リクエスト状況に柔軟に対応して最適配置が行える必要性もある．

本研究で用いるベンチマークでの 2 種類のリクエストを例として，稼働するマイクロサービスとその間での通信回数を図示したものが図 2 である．この例でも，2 つのリクエストで各マイクロサービス間の通信回数や稼働しているマイクロサービスの種類などが大きく異なり，リクエスト状況に対応した配置の重要性が読み取れる．また，この問題は組合せ最適化問題の一種であり NP 困難に属するため計算量の問題も存在する．

3. 関連研究

3.1 マイクロサービスに関する研究

マイクロサービスアーキテクチャの主たる利点である，疎結合性による耐障害性，スケール性，新技術採用・アジャイル開発の容易性などに関する研究は進んでいる [3][4][5]．

一方でアプリケーション性能の向上に関する研究は少ない．ヘテロ構成の複数マシンへのマイクロサービス最適配置によりアプリケーションの性能を保った省電力化 [6]，リクエストに対してマイクロサービスが構成するチェーンを考慮した負荷分散によるレイテンシ削減 [7]，各マイクロサービス内でリクエストの処理順を調整することによるスループット向上 [8] 等の研究があるものの，いずれの研究も上記の問題を直接解決するものではなく，マイクロサービスの通信を考慮した最適配置を行なった先行研究は存在しない．

3.2 マイクロサービス以外の分野でのコンテナ最適配置に関する研究

マイクロサービス以外の分野ではコンテナ最適配置に関する研究が存在する．VNF(Virtual Network Function) を利用したサービスのコンテナ最適配置に関する研究に Zhang らのもの [9] がある．この研究では複数種類のアプリケーションがサーバーに同時にデプロイされた状況を仮定し，リソース配分によるスループット向上と物理マシン間通信削減によるレイテンシの改善，双方を考慮したコンテナ最適配置を行っている．

通信レイテンシを減らすためには同一アプリケーションに属するコンテナを同一マシンに配置するべきである．しかしながら，同一アプリケーションに属するコンテナは同一のリソースを多く使用するため，同一マシンに配置するとリソースが枯渇しスループットが低下する．

Zhang らの研究ではリソース使用率のマシン間分散と，同一アプリケーションに属するコンテナが異なるマシンにまたがって存在すること，それぞれに対してコスト関数を設け，その線形和を最小化するように問題を定式化し，最適配置の近似アルゴリズムを提案している．

しかしながら，この研究はそもそもマイクロサービスを対象にしたものではなく，具体的に以下のような点でマイクロサービスに適さない．

- 通信に関するコスト関数が，同一アプリケーションのコンテナが物理マシンをまたいで存在する場合に画一的にかけるコストとなっており，アプリケーション内でもコンテナ対 (マイクロサービス対) ごとに通信の回数や容量にばらつきがあることを考慮していない．このモデルをそのまま適用すると，例えば 1 秒間に 100 回の通信が行われるマイクロサービス対と 1 回しか通信が行われないマイクロサービス対が通信コスト上で同じ扱いを受けるため，通信レイテンシが最大限に削減されない
- 上記のマイクロサービス対ごとに異なる通信コストや，各マイクロサービスのリソース使用量測定のためのシステム実装に関する記述が存在しない．実用性の保障のためにもこの記述は必要である
- マイクロサービスに対する性能評価が存在しない

3.3 本研究の目的とアプローチ

リソース使用量の配分と物理マシン間の通信削減の双方を考慮したマイクロサービスの物理マシンへの最適配置，それによるアプリケーションのスループット向上，レイテンシ削減を本研究の目的とする．

アプローチとして Zhang らのアルゴリズムをマイクロサービスアーキテクチャに適したものに発展させることを提案する．Zhang らのアルゴリズムは数千のコンテナ数で実行可能な近似アルゴリズムであり，百度 (Baidu) の実環境においてスループットの向上，レスポンスタイムの高速化に成功した実績がある．

本研究ではこのアルゴリズムの通信に対するコスト関数をマイクロサービス対ごとに異なる通信頻度・容量のばらつきを考慮したものに変更する．また，既存のマイクロサービスアプリケーションにおいても容易に導入可能な

最適化システムを構築し，ベンチマークとなるアプリケーションを用いて最適化アルゴリズムの性能評価を行った．

4. マイクロサービス配置最適化

本章ではマイクロサービス配置最適化の理論について説明する．4.1 でマイクロサービスアプリケーションの通信を考慮したグラフによるモデル化を提案する．4.2 では Zhang のアルゴリズムを元として通信コスト部分をマイクロサービスに対応させた最適化問題の定式化を行う．4.3 では同じく Zhang のアルゴリズムを基本とした最適化アルゴリズムを説明する．

4.1 マイクロサービスアプリケーションのグラフモデル化

マイクロサービスのアプリケーションを各マイクロサービスをノード，マイクロサービス間の通信をエッジとして無向グラフ $G(V,E)$ でモデル化する．ここで

V : マイクロサービス集合

E : マイクロサービス間の通信

$w(u,v)$: マイクロサービス u,v 間の通信コスト

とする．

辺の重みとなる通信コストは以下のように最適化対象のアプリケーションのログから計算し，全体の重みの和が 1 となるように正規化を行う．リクエストによって通信経路は様々であるが，最適化したい対象の一定期間のログからリクエストの種類を問わず $\mathrm{cnt}(u,v), t(u,v)$ を集計するため，あらゆるアプリケーション，リクエスト状況に対応できる．マイクロサービス間の通信には方向性があるが，ここでは方向性は考慮せず無向グラフとして扱う．必要なログは Zipkin[10] などの分散トレーシング技術を利用することでマイクロサービスアプリケーションにおいて容易に取得・計算可能である．

$$w_{raw}(u,v) = \mathrm{cnt}(u,v) * t(u,v)$$

$$w(u,v) = \frac{w_{raw}(u,v)}{\sum_{(u,v)\in E} w_{raw}(u,v)}$$

$\mathrm{cnt}(u,v)$: u,v 間の通信回数

$t(u,v)$: u,v 間で 1 回の通信時間の 90percentile 値

グラフの例として，評価実験に用いるベンチマークアプリケーションのログでのグラフを図 3 に示す．

4.2 問題の定式化

目的はリソース使用を複数のマシンに分配しつつ物理マシン間での通信を削減することである．そこで，以下のようにリソース使用と通信それぞれにコスト関数を設け，その重み付き和を最小化するというように問題を定式化する．この定式化は Zhang[9] の方法を基本としており，$Ccost$ のマイクロサービスに適した形への変更が本研究での提案である．$Ucost$，$Ccost$ のそれぞれは以下で説明する．w_U および w_C はハイパーパラメータである．

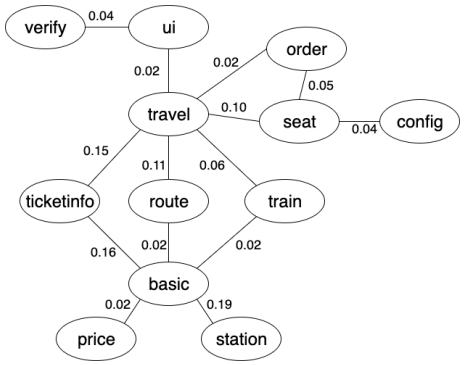


図 3 ベンチマーク TrainTicket，BasicSearch でのグラフ例

$$\min \quad w_U * Ucost + w_C * Ccost \tag{1}$$

$Ucost$: リソース使用率の分散に関するコスト (Utilization)

$Ccost$: 通信に関するコスト (Communication)

リソース使用率の分散に関するコスト

リソース使用率に関するコストは以下のように，各リソースの使用率のマシン間分散の和として定める．

$$Ucost = \sum_{r\in R}\sum_{h\in\mathbb{H}} \frac{[U(h,r) - \bar{U}(r)]^2}{|\mathbb{H}|} \tag{2}$$

R : リソース集合

$\mathbb{H}$: ホストマシン集合

$U(h,r)$: ホストマシン h におけるリソース r の使用率

通信に関するコスト

通信に関するコストは，通信のグラフにおいて物理マシンをまたぐ通信の重みの割合として定める．Zhang のアルゴリズムではこの部分を 2 つのマシンに対する関数として定めているが，本研究においてはマイクロサービス対ごとに定め，よりマイクロサービスアーキテクチャに対応したものとなっている．

$$Ccost = \sum_{(u,v)\in E} w(u,v) f(u,v) \tag{3}$$

$$f(u,v) = \begin{cases} 1 & \text{if} \quad H(u) = H(v) \\ 0 & \text{else} \end{cases}$$

E : 通信グラフの辺集合

$H(u)$: マイクロサービス u のホストマシン

制約条件

マシンの許容量を超えるリソース使用は禁止される．

$$U(h,r) \le Capacity(h,r), \forall h \in \mathbb{H}, \forall r \in R \tag{4}$$

4.3 最適化アルゴリズム

上記の最小化を実現する最適配置は組合せ最適化問題であり NP 困難となる．そこで，以下のような Zhang の貪欲アルゴリズムを基本とした近似解法を利用する [9]．Shift 操作における通信を優先する探索の追加が，アルゴリズム部分での本研究における提案である．

操作

Zhang のアルゴリズムは適当な初期配置から 1 ステップごとに最適に近い操作を選択して配置を更新していく貪欲法である．

Search

アルゴリズムにおいて実際に行う操作は以下で論ずる `Shift`，`Swap`，`Replace` のいずれかである．`Search` はそれらの中から適切な操作を選択・実行する．それぞれの操作でどのマイクロサービスを移動させるかの探索では，局所最適解を回避するため，コスト関数が最小となる 1 通りの移動をそのまま返すのではなく，最小となる 3 通りの移動の中から 1 つをランダムに選択して返す．

Shift

単純に 1 つのマイクロサービスをマシンからマシンへ移動させる操作が `Shift` である．全探索するとマシン数 M に対して $O(MV)$ の計算量がかかるが，探索候補を適当に制限することで計算量を削減する．

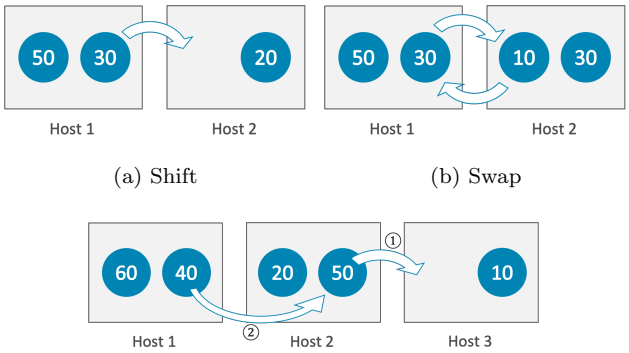
本研究で用いる手法では，リソース使用量の最も多い δM 台のマシンから最も少ない δM 台のマシンへの `Shift` のみを許容することで 1 度の探索における計算量の期待値を $O(\delta MV)$ まで改善する．また，$Ccost$ の改善が大きい 3 種類の `Shift` の探索も行う．常に各マイクロサービスが各マシンに `Shift` された場合の $Ccost$ の改善量を保持し更新することで，更新可能な優先度付きキューを利用して 1 度の探索の計算量期待値は $O(\nu \log(MV))$ となる．ν は各ノードの次数の期待値である．

Swap

`Swap` 操作では 2 つのホストマシン間でマイクロサービスを交換する．全探索すると $O(V^2)$ の計算量がかかるためこちらも適当に探索候補を絞ることで計算量を削減する．

Replace

`Replace` 操作では，`Shift` および `Swap` の探索で得られたそれぞれの移動を仮定した上で，`Shift` や `Swap` で空いたスペースに他のサービスを入れる探索を行う．つまり，あるサービスを h_2 から h_3 に移動させる際にさらに h_1 から h_2 への移動を考える．例えば図 4(c) の状況では①の移動を仮定して②の移動を採用している．単純な `Shift` 操作のみでは Host1 から Host3 へリソース使用率 40%のサービスが移動してしまう．Replace の探索についてもリソース使用率の高いマシンからの移動のみを探索するなどの制約により計算量を削減できる．



(a) Shift (b) Swap

(c) Replace

図 4 各操作の概念図

Algorithm 1 Search

Input: $P_{start}, threshold$ // 初期配置，閾値
Output: P_{best}

```
P_crt ← P_start
P_best ← P_start
while Cost(P_crt) > threshold do
  P_shift ← shiftSearch(P_crt)
  if Cost(P_shift) < Cost(P_best) then
    P_best ← P_shift
  end if
  P_swap ← swapSearch(P_crt)
  if Cost(P_swap) < Cost(P_best) then
    P_best ← P_swap
  end if
  h ← Shift, Swap でサービスがそこから離脱したマシン
  for h ∈ h do
    P_reploce ← replaceSearch(P_crt, h)
    if Cost(P_replace) < Cost(P_best) then
      P_best ← P_replace
    end if
  end for
  P_crt ← P_best
end while
```

解の精度

このアルゴリズムで得られた解が最適解の $(1+\epsilon, \theta)$ 近似であることが示されている [9]．

擬似コード

代表的に，`Search`，`shiftSearch` の擬似コードを以下 Algorithm1，2 に示す．

5. 評価環境の実装

5.1 全体構成

評価環境に求められる主な要件は以下である．

(1) マイクロサービスが分散システム上で動作する
(2) 各リクエストに対して，リクエストの処理時刻，各サービス内での処理時間・各サービス間の通信時間のログが全て取得可能である
(3) 各サービスでのリソース使用量ログが取得可能である
(4) アプリケーションに対して設定した頻度で自動的にリクエストを行い負荷をかけることができる

Algorithm 2 shiftSearch

Input: P
  $H_{hot} \leftarrow$ リソース使用率上位δM 台のマシン
  $H_{cold} \leftarrow$ リソース使用率下位δM 台のマシン
  $S \leftarrow H_{hot}$に属するサービス
  for $s \in S$ **do**
    for $h \in H_{cold}$ **do**
      $P_{tmp} \leftarrow P$ から s を h に移動させた場合の配置
      $Cost(P_{tmp})$ が Best3 に入る配置を保存していく
    end for
  end for
  $P_{ret} \leftarrow$ Best3 に入る配置のうちの 1 つをランダムに選択
  if $Cost(P_{ret}) < Cost(P)$ **then**
    return P_{ret}
  else
    return P
  end if

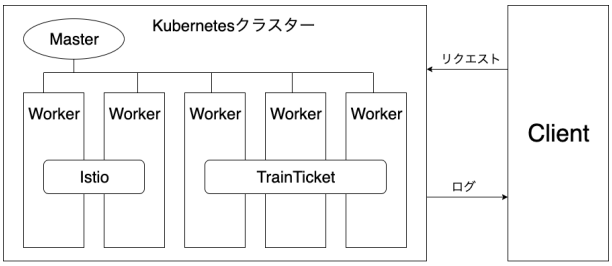


図 5　実験システムの概要図

これらを実現するため，図 5 のシステムを実装した．システムにおいて利用しているソフトウェアは基本的にオープンソースのものであり，既存のアプリケーションに容易に組み込むことが可能である．

5.2　Kubernetes クラスタ

サーバでは Kubernetes クラスタ [11] を構成し，ベンチマークのアプリケーションおよびサービスメッシュ Istio[12][13] を動作させる．アプリケーションを構成する各マイクロサービスは複数の Worker マシンに分散して配置される．

サービスメッシュ

サービスメッシュおよび分散トレーシング[*1]を利用して各リクエストの処理やリソース使用に関するログを取得可能である．Istio と連携して動作するソフトウェアとして分散トレーシングに Zipkin[10]，リソース使用量のモニタリングに Prometheus[14] を利用した．

5.3　クライアント

クライアントからは自動的にリクエストが送信され，アプリケーションに負荷をかけることができる．自動的なリクエストの作成には Selenium を利用した．

[*1] 分散されたシステムで処理されるリクエストをトレーシングする技術．サービス間の依存関係やレイテンシを解析するために用いられる

5.4　マシン環境

有線ネットワークで接続された 7 台同一構成の物理マシンを用意し，6 台を Kubernetes クラスタ，1 台をクライアントマシンとした．Kubernetes クラスタについては各物理マシン上に 1 つずつ VM(Virtual Machine) を Linux KVM によって設置し，その上で稼働させた．VM を利用した主な理由はマシン性能の変更の容易性である．VM 間通信のオーバーヘッドは物理マシン間通信のオーバーヘッドと比較して非常に小さい [2] ため，VM を中継することで通信性能に大きい差は生じない．表 1，2 に各物理マシンおよび VM のスペック，用途を記載する．

表 1　物理マシンスペック

CPU	Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
メモリ	64GB
OS	Ubuntu 18.04.3 LTS 64bit

表 2　各 VM の用途およびスペック

VM 名	CPU コア	メモリ	用途
master	20	8GB	Kubernetes master
mesh1〜2	20	16GB	Istio 動作
worker1〜3	8	16GB	TrainTicket 動作

6.　最適配置の評価

6.1　実験目的・概要

実験の目的は提案したアルゴリズムによって計算された最適配置でのアプリケーションのスループットおよびレイテンシ改善の評価である．一般に，アプリケーションにかける負荷を変化させるとアプリケーションのスループットを超える点で急激に応答時間が悪化する，または正常な処理が行われなくなる．この変化点の提案手法による最適配置と比較手法による配置との間での差から，最適配置でのスループット向上を評価した．また，特定の負荷下での応答時間の各配置間での差からレイテンシの改善について評価を行った．

6.2　ベンチマーク

ベンチマーク用アプリケーション TrainTicket[15] を利用した．TrainTicket は 40 種のマイクロサービスによって構成される，鉄道路線の経路や運賃の検索・鉄道の予約等をする Web サービスを模したアプリケーションである．

アルゴリズムの汎用性検証のため 2 種類のリクエストで評価を行った．BasicSearch は始発駅と終着駅のみを指定するシンプルな経路探索リクエスト，AdvancedSearch はさらに最安値のものを検索するなどより CPU 負荷が重く，多数のマイクロサービスを経由するリクエストである．図 2 にそれぞれのリクエストの辿る経路を記述している．

6.3 プロファイル情報の取得

最適配置計算のための各サービスのリソース使用量，通信・実行時間ログ取得を目的に，予備実験として以下の表3のようなそれぞれの負荷をアプリケーションにかけて，それらのログを取得した．この際のマイクロサービス配置はKubernetesデフォルトスケジューラによるものを使用した．

表 3 予備実験条件

リクエスト	インターバル [s]	回数	備考
BasicSearch	0.4	200	比較的軽い路線検索
AdvancedSearch	2	100	比較的重い路線検索

取得したログから，第2章のアルゴリズムに従い，表4に示す複数の重みで最適配置を計算した．最適配置はBasicSearch，AdvancedSearchの双方に対して別々に計算を行なった．重みそのものの最適化は行なっていない．

今回の実験においてはベンチマークでのディスクI/Oが少ないため，最適化対象のリソースはCPUとメモリのみとした．また，分配するリソース量は各マイクロサービスが使用したリソース量のみであり，各ホストマシン内で動作するデーモンプロセスなどの使用するリソース量は全く計算に入れていない．

表 4 最適化のためのパラメータ設定

割当名	w_U	w_C	備考
Network	1	1	通信を重視した割当
Utilization	1	0	リソース配分のみを考慮した割当
Both	100	1	通信とリソース配分双方を重視した割当

6.4 実験内容

実験条件

計算された配置および比較手法による配置でベンチマークをデプロイし，リクエストのインターバルを変更することで負荷を変え，各負荷下での総応答時間を配置間で比較した．実験条件は表5の通りである．アプリケーションは60s応答がない場合タイムアウトする設定となっており，全体の20%以上のリクエストでタイムアウトやエラーのために正常な応答がない場合は実行不可能と判定した．

表 5 実験条件

リクエスト	インターバル [s]	回数
BasicSearch	0.4s〜0.6s まで 0.05s ごと	200
AdvancedSearch	1.8s〜2.2s まで 0.1s ごと	100

比較手法

比較対象にKubernetesデフォルトスケジューラによる配置を利用した．Kubernetesはマイクロサービス動作プラットフォームのデファクトスタンダードであるため比較手法として妥当である．Kubernetesスケージューラではアプリケーションの開発者が事前に設定したリソースのリクエスト量，リミット量や各ホストマシンの割当時のリソース消費量を考慮してスケジューリングを行うが，実際の各マイクロサービスのリソース消費量や通信等のログは全く使用しない [16]．

最適配置ではKubernetesデフォルトの配置と比較してスループット，レイテンシ共に改善が期待される．

6.5 実験結果

リソース使用率分配とスループット向上に関する評価

実験における応答の90percentile時間を各リクエスト，各負荷で描画したものが図6である．プロットのない部分はその配置でタイムアウト等により実行が不可能であったことを示す．

リソースの配分を重視しているパラメータ設定のUtilizationとBothでは他の配置で実行不可能になっている負荷の下でも比較的短時間で応答が返っており，スループットの向上に成功している．UtilizationとBothの間の差はBasicSearchではほぼ見られないが，よりCPU負荷の重いリクエストであるAdvancedSearchではよりリソース配分を重視したUtilizationに軍配が上がっている．

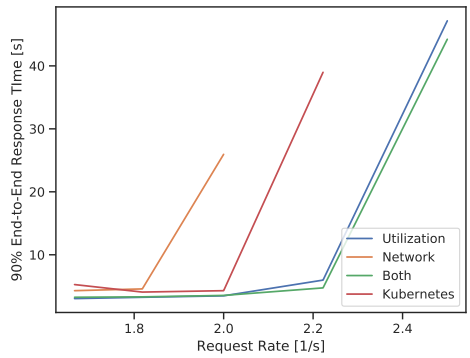
また，代表的に特定条件下での各マシンに配置されたマイクロサービスによるCPU使用率およびメモリ使用率の合計を示したものが図7である．メモリ使用率はKubernetesデフォルトを除く3つの配置で概ねよく配分されている．CPU使用率はUtilizationとBothで他の配置と比較してうまく配分されている．

レイテンシ削減に関する評価

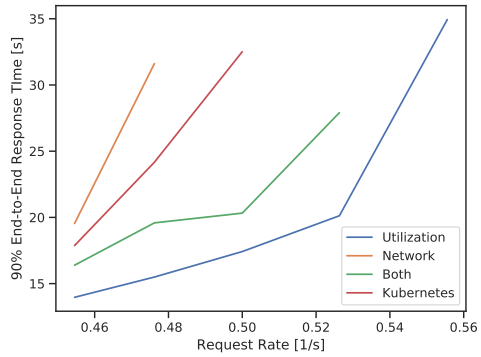
リソースの配分および通信の削減によるレイテンシの削減について，AdvancedSearchを例にとって評価する．まず以下の図8，9に特定の負荷下におけるそれぞれの配置での各リクエストに対するend-to-end応答時間，マイクロサービス内部実行時間の合計，マイクロサービス間通信時間の合計それぞれの分布を示す．箱ひげ図の上下のひげの先端はそれぞれ90%，10%点，箱の端は75%，25%点とした．8は比較的負荷が軽い場合，図9は比較的負荷が重い場合の代表例である．

本研究のベンチマークは内部実行時間の占める割合が大きいものであるため，総応答時間と内部実行時間の分布はほぼ一致している．これらの時間はUtilizationおよびBothの，リソース使用量配分を意識した配置で削減ができており，特にUtilizationで改善が大きい．

一方，通信時間は全体の応答時間に与える影響は小さい

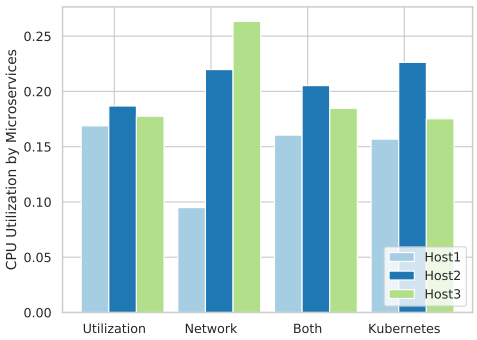


(a) BasicSearch

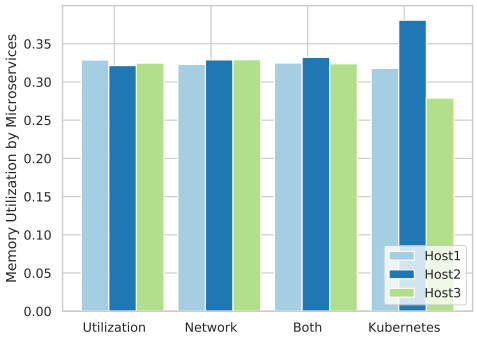


(b) AdvancedSearch

図 6 各配置におけるアプリケーションの負荷耐性



(a) CPU 使用率



(b) メモリ使用率

図 7 AdvancedSearch，2.2s 間隔リクエスト下での 3 つのマシンのリソース使用率

ものの，Network や Both などの通信時間に焦点を置いた配置で大きく削減できていることがわかる．例えば図 8 では，通信時間合計の 90percentile 値で，Network や Both の配置で Kubernetes デフォルトと比較して 20%以上の削減を果たしている．

内部実行時間，通信時間ともに中央値付近での変化はあまりなく，おおよそ 80percentile 以降などの部分での悪化が著しい．内部実行時間では CPU などのリソースが枯渇している部分だと考えられ，リソース使用量分配の重要性がわかる．通信時間部分はマシンのネットワーク I/O スループットを超える I/O 要求や，回線が混雑する場合の TCP/IP 再送制御などが関わっていると考えられ，こちらもマシン間通信の削減の重要性を強調する結果と言える．

6.6 考察

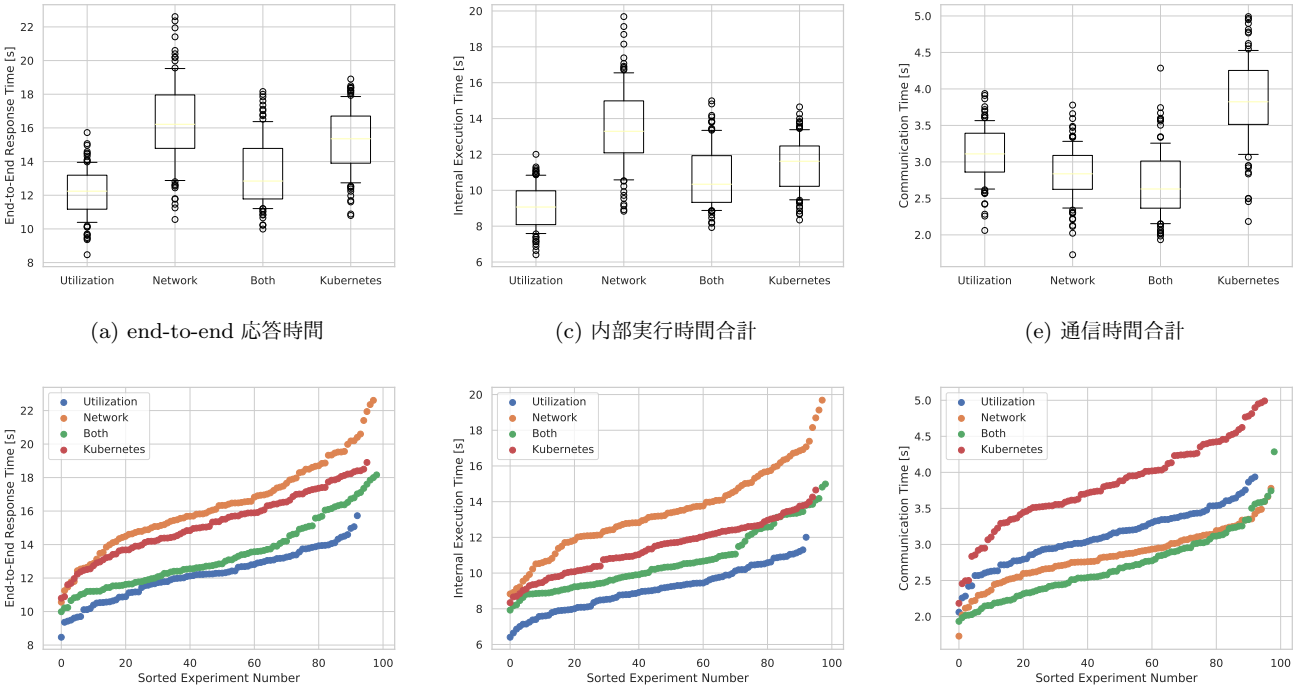
それぞれの配置で目的とするリソース使用率の配分，通信時間の削減に成功し，スループットの向上やレイテンシの改善を果たした．

今回扱ったベンチマークでは内部実行時間が支配的であるため，通信時間の削減の end-to-end の応答時間への影響は小さく，リソース使用率の配分を重視する Utilization でのレイテンシ改善が大きかったものの，通信時間の占める割合がより大きくなるアプリケーションでは Both のようなパラメータ設定での改善が大きくなるものと考えられる．

図 6 と図 7 を比較すると，マイクロサービスによる CPU 使用率の配分がうまく行えるとスループットが向上するという一定の関係性が伺える．これはアプリケーションが CPU 負荷が支配的となる特性を持っていたためであり，例えばディスク I/O が多いアプリケーションの場合はディスク I/O の配分が重要になるものと考えられる．

アプリケーションのボトルネック分析により $Ucost$ 内での各リソースの使用率分散に重み付けを行い，ボトルネックとなるリソースがよりうまく配分されるように調整することで，よりアプリケーションの特性に合ったリソース配分が可能になるだろう．

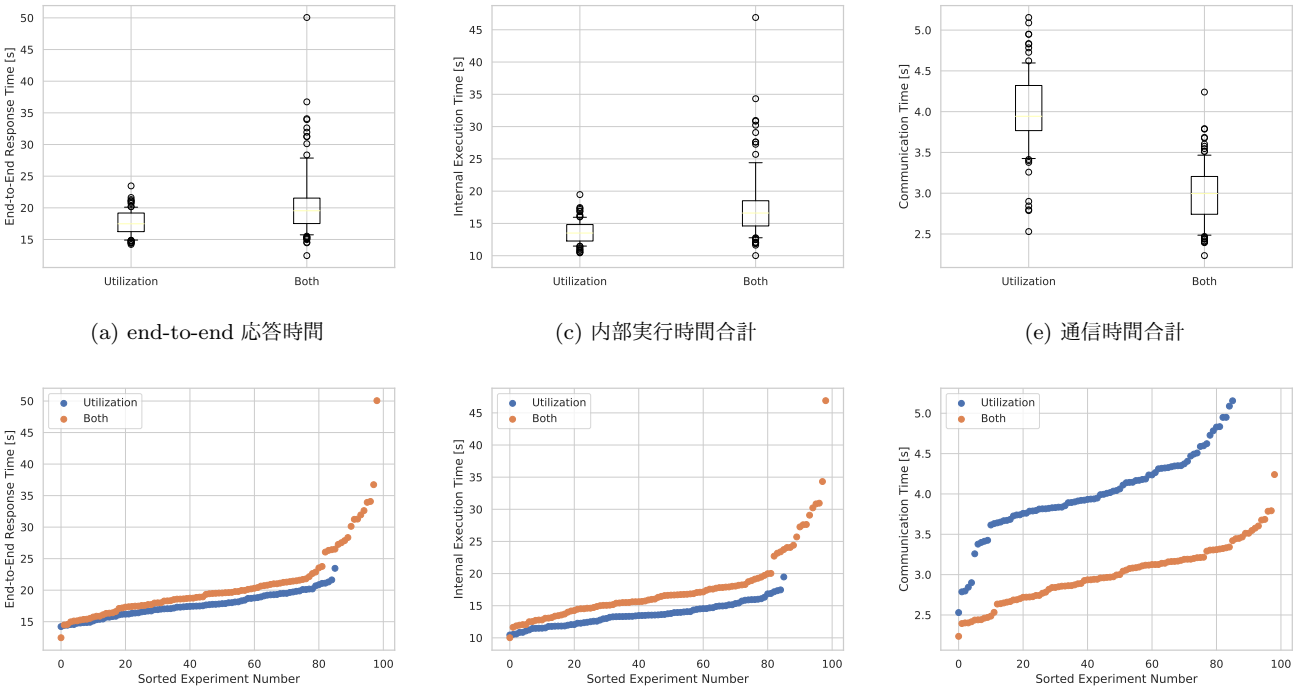
最適化計算では Utilization では 3 台のマシンでほぼ同じリソース使用率となる予測だったが，図 7 では CPU 使用率に多少のブレがある．予備リクエスト時や評価実験時のノイズによるぶれは考えられ，予備リクエストの期間や回数を増やすことでより正確な最適配置，実験回数を増やすことでより正確な評価が可能である．最適配置に関しては実用時にも同様のことが言え，より多くのログを使用して最適化計算を行えばより正確な最適配置が可能である．ただし，余計に長期的なログを使用するとその区間で平滑化された最適配置のようなものが計算されることになり，結果的に瞬間的な高負荷がかかる場合などに対する最適化は行えないので注意が必要である．特にスループット向上を



(a) end-to-end 応答時間　(c) 内部実行時間合計　(e) 通信時間合計

(b) end-to-end 応答時間詳細 (Sorted)　(d) 内部実行時間合計詳細 (Sorted)　(f) 通信時間合計詳細 (Sorted)

図 8　AdvancedSearch，2.2s 間隔での内部実行時間，通信時間，end-to-end 応答時間



(a) end-to-end 応答時間　(c) 内部実行時間合計　(e) 通信時間合計

(b) end-to-end 応答時間詳細 (Sorted)　(d) 内部実行時間合計詳細 (Sorted)　(f) 通信時間合計詳細 (Sorted)

図 9　AdvancedSearch，1.9s 間隔での内部実行時間，通信時間，end-to-end 応答時間

目的とする場合，実用時には最適化対象のログの中で負荷の大きい部分をより強く最適配置に反映するような仕組みが求められる．

7. 結論

本研究ではリソース使用量の各物理マシンへの配分と物理マシン間の通信削減の双方を考慮したマイクロサービスの最適配置によりアプリケーションのスループット・レイ

テンシ改善を目的とした．アプローチとして，Zhang のアルゴリズムの通信に対するコスト関数および最適な操作の選択をマイクロサービスアーキテクチャに対応したものに変更することを提案した．その後，ベンチマークとなるアプリケーションを用いてアルゴリズムの性能評価を行った．

結果として，提案手法による配置で Kubernetes デフォルトスケジューラと比較してリソース使用量の配分と通信の削減に成功し，スループット，レイテンシの改善を果たした．それによって，Zhang のアルゴリズムのマイクロサービスへの適用可能性と提案する通信コスト関数の有用性を示した．また，オープンソースのソフトウェアを用いた，Kubernetes により動作するマイクロサービスアプリケーションに容易に組み込むことが可能な最適化システムを構築した．

今回は物理マシン，アプリケーション共に比較的小規模な構成で実験を行なったが，実用環境に対応してより大規模な環境におけるアルゴリズムの性能評価が必要となる．また，アルゴリズムの各コストの重み w_U, w_C は今回ハイパーパラメータとして設定し，複数の組を人間の手で設定し実験を行なったが，これらの最適化も重要である．

本研究では予備実験のログから 1 度最適化をした配置による結果のみを示したが，最適配置を行なった後のログをさらに使用して再配置を動的に繰り返していく場合の性能評価も求められる．

謝辞 本研究の一部は JST さきがけ JPMJPR19M3 の助成を受けたものである．

参考文献

[1] Thönes, J.: Microservices, *IEEE software*, Vol. 32, No. 1, pp. 116–116 (2015).
[2] Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M. and Steinder, M.: Performance evaluation of microservices architectures using containers, *2015 IEEE 14th International Symposium on Network Computing and Applications*, IEEE, pp. 27–34 (2015).
[3] Newman, S.: *Building microservices: designing fine-grained systems*, ” O’Reilly Media, Inc.” (2015).
[4] Hassan, S., Ali, N. and Bahsoon, R.: Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity, (online), DOI: 10.1109/ICSA.2017.32 (2017).
[5] Zimmermann, O.: Microservices tenets, *Computer Science-Research and Development*, Vol. 32, No. 3-4, pp. 301–310 (2017).
[6] Hou, X., Liu, J., Li, C. and Guo, M.: Unleashing the Scalability Potential of Power-Constrained Data Center in the Microservice Era, *Proceedings of the 48th International Conference on Parallel Processing*, ACM, p. 10 (2019).
[7] Niu, Y., Liu, F. and Li, Z.: Load balancing across microservices, *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, pp. 198–206 (2018).
[8] Kannan, R. S., Subramanian, L., Raju, A., Ahn, J., Mars, J. and Tang, L.: GrandSLAm: Guaranteeing SLAs for jobs in microservices execution frameworks, *Proceedings of the Fourteenth EuroSys Conference 2019*, ACM, p. 34 (2019).
[9] Zhang, Y., Li, Y., Xu, K., Wang, D., Li, M., Cao, X. and Liang, Q.: A communication-aware container redistribution approach for high performance VNFs, *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 1555–1564 (2017).
[10] Zipkin: Zipkin (2019). `https://zipkin.io/`.
[11] Authors, T. K.: Production-Grade Container Orchestration (2020). `https://kubernetes.io/`.
[12] Li, W., Lemieux, Y., Gao, J., Zhao, Z. and Han, Y.: Service Mesh: Challenges, state of the art, and future research opportunities, *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, IEEE, pp. 122–1225 (2019).
[13] Authors, I.: Istio 1.4 (2019). `https://istio.io/`.
[14] Authors, P.: ”Prometheus - Monitoring system & time series database”.
[15] Zhou, X., Peng, X., Xie, T., Sun, J., Xu, C., Ji, C. and Zhao, W.: Poster: Benchmarking Microservice Systems for Software Engineering Research, *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, IEEE, pp. 323–324 (2018).
[16] Authors, T. K.: ”Kubernetes Scheduler” (2020). ”`https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/`”.