スケーラビリティ向上のためのモノリシックアーキテクチャのマイクロサービス化における自動分割の検討

伊尾木 啓太[1]　穐山 空道[1,a)]

概要：近年、利用者数の増加に伴い、既存のモノリシックアーキテクチャにスケーリングが求められることがある。しかし、モノリシックアーキテクチャは構造上スケーラビリティが低く、負荷分散が困難であるという課題を抱えている。この問題を解決する手段として、システムをマイクロサービスアーキテクチャへ移行する方法が注目されている。しかし、マイクロサービス化を行う際には、既存のコードを分割する必要があり、その決定が難しいという新たな課題が生じる。本研究では、既存のコードの解析により得られる情報から、ある分割時のボトルネック解消に必要なリソース量を評価する指標を提案する。最終的には全分割パターンについて評価を行い、最もよい分割パターンを決定することを目指す。

1. はじめに

モノリシックアーキテクチャは複数機能を 1 つアプリケーションとして構築するため、あるコンポーネントに需要増加が生じた場合でもシステム全体をスケールする必要があり、簡単にスケーリングを行えないという課題がある [1]。この課題を解決するためにモノリシックアーキテクチャのマイクロサービスアーキテクチャへの移行が検討されている [1]。マイクロサービスアーキテクチャとは、アプリケーションを複数の独立したサービスで構成するソフトウェア設計手法である。各サービスは他サービスとは独立したリソースを利用するため、スケーリングがモノリシックアーキテクチャよりも簡単である。

マイクロサービスアーキテクチャへの移行には複数の方法が存在するが、本研究ではすでに存在するモノリシックシステムを分割しマイクロサービス化する手法を考える。この手法は、求めた分け方が本当に目的に対して最適な分け方である保証がないという課題が存在する。

2. 自動分割によるマイクロサービス化

2.1 提案手法

本研究では処理増加により発生したボトルネックを最も少ないリソース追加で解決する分割方法を求める。そのために全分割パターンに対して評価指標を適用し、指標の値が最も高くなる分割パターンを探索アルゴリズムにより探索する。具体的には以下の手順で自動分割を行う。

(1) テストケースを実行し、各関数の被呼び出し回数と関数が呼び出される順序を調べる。
(2) システム内の関数の組み合わせをサービスとしたときの評価指標の値を求める。本研究ではサービスの最小単位を 1 つの関数とする。
(3) 評価指標が最大となるような分割パターンを探索アルゴリズムによって求める。

ある分割 G のときのシステム全体に必要なリソース量 $E(G)$ を以下の式で評価する。

- $E(G) = \frac{P(G)}{C(G)}$
- $P(G) = \sum_{i} \frac{1}{F_i}$
- $C(G) = \sum_{i=0}^{n} \sum_{k=0}^{i-1} S_{(i,k)}$

ここで $P(G)$ はある分割 G のときのボトルネック緩和率、$C(G)$ は通信コストである。

$P(G)$：ある分割を行った際にボトルネックがある追加リソースによりどの程度解消されるかを表す指標である。本研究ではサービス内に含まれる関数の数が少ないほど、与えられたリソースをボトルネック解消に使用できるとする。この考えに基づき、各サービスのボトルネック緩和率を分子を 1、分母をあるサービス i 内に含まれる関数の数 F_i とし、全サービスについてこの値の和を取る。

$C(G)$：ある分割を行ったときに追加で発生する通信コストを表す指標である。関数の被呼び出し回数とは、システムを動かしたときに各関数が呼び出される回数である。$C(G)$ はサービス間の通信回数の総和であり、あるサービス i が別のサービス k を呼び出す回数 $S_{(i,k)}$ を足し合わ

[1] 立命館大学 情報理工学部
[a)] s-akym@fc.ritsumei.ac.jp

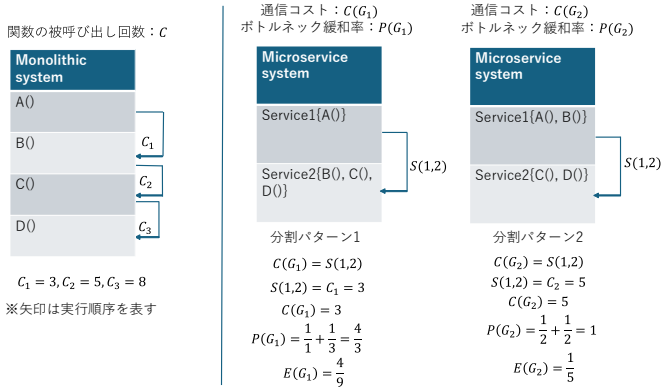

図 1: 提案手法の全体像

せる。これを関数が呼び出されれる順序と関数の被呼び出し回数を用いて求める。

図 1 に、提案手法の全体像を示す。この図では 4 つの関数があり、それらの被呼び出し回数を用いて二つの分割パターンを比較している。比較の結果分割パターン 1 が分割パターン 2 よりも優れた分け方であると判断する。

2.2 手法の適用例

提案手法を適用する具体例として、簡易的な物理エンジンを作成した。この物理エンジンはある個数の質点の動きを 10 秒間シミュレーションする。シミュレーションを高速化するため、空間を分割しそれぞれの空間内で質点の衝突判定を行う。前ステップの情報をリセットする初期化関数、質点の位置を登録する登録関数、ある位置がどの分割空間に属しているかを識別する識別関数、衝突の可能性がある質点のペアを抽出するペア抽出関数、衝突判定を行う衝突判定関数、衝突後の処理を行う反射関数、質点の半径を返す半径関数の 7 つの関数がある。

第 2.1 章で示した手順に従い、以下の方法で最良の分割を求めた。

(1) コードにログをとる関数を埋め込み、関数の被呼び出し回数を調べた。次にコードを読んで関数が呼び出される順序を調べる。

(2) 求めた情報をもとに、$E(G)$ が最大となる G を求める。本来は膨大な分割パターンの集合から探索によって $E(G)$ が最大値をとる G を求めるが、この例ではすべての G について $E(G)$ を計算した。

$E(G)$ の値が高かった上位 3 つの分割を上から順に $G1$, $G2$, $G3$ とし、表 1 に示す結果が得られた。各分割パターンのサービス内容は以下のとおりである。

$G1$ (初期化 登録 識別), (ペア抽出 衝突判定 反射 半径)
$G2$ (初期化), (登録 識別), (ペア抽出 衝突判定 反射 半径)
$G3$ (初期化), (登録 識別 ペア抽出 衝突判定 反射 半径)

表 1: 評価指標上位 3 分割パターン

分割	$P(G)$	$C(G)$	$E(G)$
$G1$	0.583333	600.000000	0.0009722222
$G2$	1.750000	60600.000000	0.0000288779
$G3$	1.166667	60000.000000	0.0000194444

3. 議論

3.1 関連研究

マイクロサービス化の自動分割の関連研究として [2] がある。本研究との違いは、マイクロサービス化の目的である。文献 [2] ではマイクロサービス化による保守の容易さと耐障害性を求めている。具体的にはサービスの規模や凝集度について評価を行い、その評価をもとに分割を行う。一方で本研究ではボトルネック解消のみを目的とする。

3.2 今後の課題

今後の課題は以下の 3 点である。

(1) 評価指標の妥当性：本研究で用いた評価指標では、リソースを追加した際のボトルネック解消率を全関数で同一としている。しかし実際には処理によってスケーラビリティは大きく異なるため、スケーラビリティの高い処理を単一のサービスにすることがボトルネック解消には有用である。このため、$C(G)$ を正規化したり、重みを付与したりする必要がある。

(2) 手法の適用可能性：本研究では、解析で得たボトルネックはマイクロサービス化した後も続けてボトルネックになりうるものであるというモデルでのみ成り立っている。つまりこの方法ではマイクロサービス化した後に発生する新たなボトルネックに対処できない。

(3) マイクロサービス化の目的：本研究ではシステムの遅延や信頼性等を考慮していない。スケーラビリティを向上させつつシステムの遅延を最小化したい場合などには異なる指標が必要である。

謝辞 本研究は、JSPS 科研費 JP23K28078 の支援を受けたものである。

参考文献

[1] Oyekunle Claudius Oyeniran, Adebunmi Okechukwu Adewusi, Adams Gbolahan Adeleke, Lucy Anthony Akwawa and Chidimma Francisca Azubuko: Microservices Architecture in Cloud-Native Applications: Design Patterns and Scalability, *International Journal of Advanced Research and Interdisciplinary Scientific Endeavours*, pp. 2107–2124 (2024).

[2] Davide Taibi and Kari Systa: A Decomposition and Metric-Based Evaluation Framework for Microservices, *Cloud Computing and Services Science - 9th International Conference, CLOSER 2019, Revised Selected Papers*, pp. 133–149 (2019).