マイクロサービスアーキテクチャにおける
通信経路最適化による応答速度向上に関する研究

21T314　芝　昌隆　（最所研究室）

1.　はじめに

デジタルトランスフォーメーション（DX）が推進されている。DX 推進には様々な試行錯誤が必要であり，情報システムの更新が容易なマイクロサービスアーキテクチャが適する。しかし，多くの細かく独立したサービスのインスタンスを組み合わせてアプリケーションを構築するため，特定のインスタンスの遅延により，アプリケーションの応答時間が長くなる可能性がある。本稿は，通信経路を最適化して応答速度を向上するシステムを提案する。

2.　マイクロサービスアーキテクチャと課題

従来の情報システムは，多数の機能を 1 つのソフトウェアに纏めてアプリケーションを構築する。一方，マイクロサービスアーキテクチャを採用する情報システムは，図 1 に示すように，最小の機能を提供するサービス(図では$S1, S2, S3$)を組み合わせて 1 つのアプリケーションを構築する。1 つのサービスは同じサービスを提供する複数のインスタンス($S1$では$I_{1,1}, I_{1,2}, I_{1,3}$)で行う。それらサービス毎の機能開発やインスタンス毎の起動・停止により，試行錯誤が容易になる。

アプリケーションを構成するインスタンス群の通信を監視・制御するために，サービスメッシュが使用される。サービスメッシュはプロキシを用いて，インスタンス群の通信を監視し，リクエストの送信先インスタンスを制御する。リクエストの振り分け方式には，ラウンドロビン方式や接続数の少ないインスタンスに振り分ける最小接続方式がある。さらにそれらの方式には，インスタンスの処理能力に応じて優先度を重みとして設定できる。しかし，インスタンス間の応答時間を考慮しておらず，アプリケーションの応答時間が長くなる場合がある。

この問題を解決するために，インスタンス間の応答時間に基づいてインスタンス間に重みを設定するシステムを提案する。

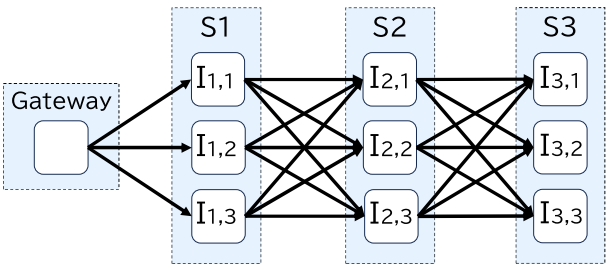

図 1 Web アプリケーションの構成

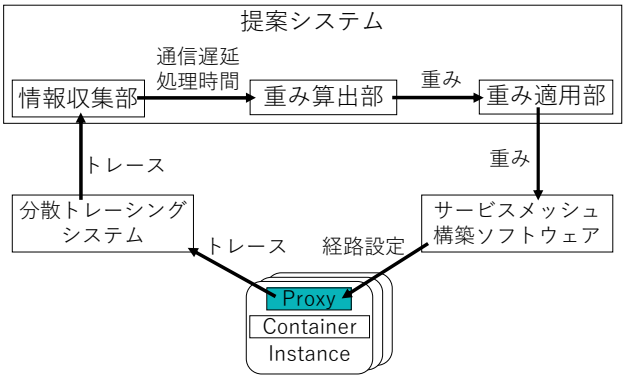

図 2 システム構成

3.　提案システム

提案システムの構成を図 2 に示す。提案システムは，情報収集部，重み算出部，重み適用部から成る。

(1) 情報収集部

情報収集部は，リクエストの流れを時系列で追跡する分散トレーシングシステムから，インスタンス間の通信遅延とインスタンス毎の処理時間を含むトレースを取得する。

(2) 重み算出部

重み算出部は，情報収集部で取得した情報に基づき，インスタンス間に設定する重みを決定する。

アプリケーションを構成するサービスの数をN，k番目のサービスに属するインスタンスの数をM_k，k番目のサービスに属するl番目のインスタンスを$I_{k,l}$とする。また，2 つのインスタンスI_{k_1,l_1}，I_{k_2,l_2}間の通信遅延を$L(I_{k_1,l_1}, I_{k_2,l_2})$，インスタンス$I_{k_1,l_1}$の処理時間を$P(I_{k_1,l_1})$とする。

$x = N$のとき，インスタンス$I_{x,i}$の実行時間$E(I_{x,i})$を式(1)で算出する。

$$E(I_{x,i}) = P(I_{x,i}) \tag{1}$$

$x < N$のとき，2 つのインスタンス$I_{x,i}$，$I_{x+1,j}$間の応答時間$R(I_{x,i}, I_{x+1,j})$と重み$W(I_{x,i}, I_{x+1,j})$をそれぞれ式(2)，式(3)で算出する。

$$R(I_{x,i}, I_{x+1,j}) = L(I_{x,i}, I_{x+1,j}) + E(I_{x+1,j}) \tag{2}$$

$$W(I_{x,i}, I_{x+1,j}) = \frac{R(I_{x,i}, I_{x+1,j})^{-1}}{\sum_{k=1}^{M_{x+1}} R(I_{x,i}, I_{x+1,k})^{-1}} \tag{3}$$

さらに，$I_{x,i}$の実行時間$E(I_{x,i})$を式(4)で算出する。

$$E(I_{x,i}) = P(I_{x,i}) + \frac{M_{x+1}}{\sum_{k=1}^{M_{x+1}} R(I_{x,i}, I_{x+1,k})^{-1}} \quad (4)$$

(3) 重み適用部

重み適用部は重み算出部で算出したインスタンス間の重みを，サービスメッシュ構築ソフトウェアを介してプロキシに適用する。

(4) 提案システムの実装

マイクロサービスアーキテクチャの基盤として Kubernetes[1]を用いた。また，トレースを収集・取得するための分散トレーシングシステムとして Jaeger[2]を用いた。サービスメッシュの構築には Istio[3]を用いた。

4. 評価

重み付け無しの場合と提案システムによる重み付けを適用した場合で，同様のリクエストを送信した際のアプリケーションの応答時間を比較し，応答速度を確認する。

4.1. 評価方法

図 1 に示す 3 つのサービス S1, S2, S3 からなるアプリケーションを Kubernetes 基盤上に実装して評価した。サービス S1～S3 は，それぞれ 3 つのインスタンスによって提供される。また，S1 が S2 を，S2 が S3 を呼び出す。

リクエストの振り分け方式は Istio のデフォルト設定である最小接続方式とする。重み付け無しの場合とインスタンス間に提案システムによる重み付けを適用した場合で，それぞれ 1,000 件の同一リクエストを一斉に送信し，アプリケーションの応答時間とインスタンスの処理時間，インスタンス間の通信遅延を計測する。

4.2. 評価結果と考察

重み付けの有無それぞれにおける，アプリケーションの応答時間のヒストグラムを図 3 に示す。重み付けによって平均応答時間が 168ms から 104ms へ短縮した。さらに，従来は 400ms を超える応答があったのに対して，重み付けした場合の応答時間は全て 300ms 以下となった。

重み付けの有無それぞれにおける，サービス S1 の 3 つのインスタンスの処理時間を図 4 に示す。重み付け無しの場合は，同じサービスに属するインスタンス同士の処理時間の差が大きい。一方，重み付けした場合は，インスタンス同士の処理時間の差が減少している。インスタンスの処理時間を考慮してリクエストを振り分けることにより適切にリクエストが分散されたと考えられる。その結果，図 3 に示すようにアプリケーションの応答時間が減少したと考えられる。

インスタンス間の通信遅延にもインスタンス

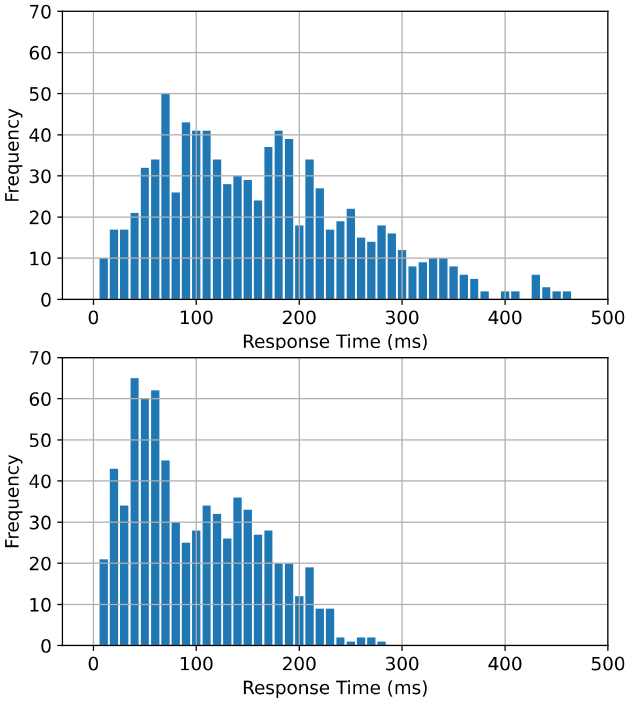

図 3 応答時間の分布の変化(上が重み付け無し)

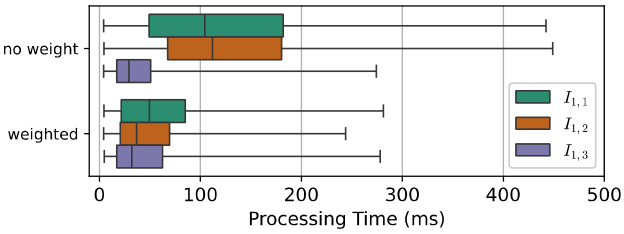

図 4 S1 のインスタンスの処理時間の変化

の処理時間と同様の傾向が見られた。しかし，平均処理時間が 91ms. であるのに対して平均通信遅延は 2ms. と極めて小さく，処理時間が支配的であると考えられる。

5. おわりに

本稿では，マイクロサービスアーキテクチャにおけるアプリケーションの応答速度向上を目的として，インスタンス間の応答時間を考慮した経路制御を実現するシステムを提案して実装評価した。その結果，アプリケーションの平均応答時間だけでなく，最遅応答時間が短縮することを確認した。

参考文献

[1] Kubernetes, https://kubernetes.io(2025 年 1 月 4 日参照)
[2] Jaeger, https://www.jaegertracing.io(2025 年 1 月 4 日参照)
[3] Istio, https://istio.io(2025 年 1 月 4 日参照)