マイクロサービス・アーキテクチャにおける 通信情報を用いた呼び出し関係可視化手法

北島　信哉†　　松岡　直樹†

† 株式会社富士通研究所 ソフトウェア研究所 新世代ＩａａＳ・ＰａａＳプロジェクト
〒 211–8588 川崎市中原区上小田中 4-1-1
E-mail: †{kitajima.shinya,matsuoka.naoki}@jp.fujitsu.com

あらまし　近年，マイクロサービス・アーキテクチャと呼ばれる Web サービス設計手法が注目を集めている．マイクロサービス・アーキテクチャはさまざまなメリットをもたらすが，障害分析が複雑になるというデメリットもあわせもっている．Zipkin のような分散トレーシングシステムも存在するが，サービスのソースコードの改変が難しい企業システムや大規模 Web サービスに適用することは困難である．そこで本稿では，各サーバで取得できる通信に関する情報をもとにマイクロサービス間の呼び出し関係を推測し，可視化する手法を提案する．提案手法を用いることにより，ソースコードを改変することなくマイクロサービス間の呼び出し関係が可視化できるため，マイクロサービス・アーキテクチャにおける障害分析が行いやすくなる．実際に稼働しているマイクロサービス・アーキテクチャのシステムにおいて通信ログを取得し，評価を行った結果から，提案手法を用いることでマイクロサービス間の呼び出し関係を高い精度で可視化できることを確認した．

キーワード　マイクロサービス・アーキテクチャ，分散システム，可視化，通信ログ，フロー

1. はじめに

近年，マイクロサービス・アーキテクチャと呼ばれる Web サービス設計手法が注目を集めている．マイクロサービス・アーキテクチャでは，1 つの Web サービスを細かい粒度のサービス（マイクロサービス）に分割し，マイクロサービス同士が Web API のようなシンプルな手段によって疎結合に連携することで，1 つの Web サービスを構成する [6]．元々は，Amazon や Netflix といった巨大な Web サービスをもつ企業から自然発生的に生まれた手法であり，James Lewis らによって，この手法は Microservices（マイクロサービス・アーキテクチャ）として再定義された [4].

マイクロサービス・アーキテクチャがもたらすメリットは，技術の多様性，回復性，スケーリングやデプロイの容易性など，多数挙げられる．一方で，分散システム化することによるデメリットも複数存在する．デメリットの 1 つに，障害分析の複雑さが挙げられる．巨大な Web サービスを構成するマイクロサービスの数は数十，数百にのぼり，処理遅延やエラーが発生した場合，その中から原因箇所を見つけ出すのは非常に困難である．

このような問題を解決するための 1 つの手段として，2012 年に Twitter, Inc. がオープンソースプロジェクトとして発表した Zipkin [8] が挙げられる．Zipkin は 2010 年に Google Inc. が発表した Dapper [9] と呼ばれる分散トレーシングシステムをもとにしている．Zipkin を利用することで，マイクロサービス間の呼び出し関係や各マイクロサービスにおける処理時間を可視化し，処理遅延の原因となっている箇所を見つけ出すことができる．

しかし，Zipkin を利用するためには，マイクロサービス間で通信を行う際に ID の受け渡しを行うとともに，Zipkin サーバに対してログを送信するように各マイクロサービスのソースコードを変更する必要がある．そのため，既存のサービスのコードを容易に変更できないような企業システムや，すでに稼働中の大規模 Web サービスにおいては，Zipkin を利用するための障壁が大きい．

そこで本稿では，各サーバで取得できる通信情報（送信元 IP アドレス，送信先 IP アドレス，送信先ポート番号，送信時刻，受信時刻）をもとにマイクロサービス間の呼び出し関係を推測し，可視化する手法を提案する．提案手法では，ある通信に対して，その通信（子通信）が発生するきっかけとなった通信（親通信）が存在すると仮定し，子通信の送信時刻からさかのぼって閾値以内の時間に受信した通信を親候補通信とする．この処理を一定量の通信ログに対しておこない，親候補通信の中から不要なものを削除し，残ったものを親通信と推測する．提案手法を用いることにより，例えばあるマイクロサービス・アーキテクチャの Web サービスにおいて処理遅延が発生した際，原因となっている可能性のあるマイクロサービスを絞り込むことができる．

以下，2. で想定環境について述べ，3. で提案手法について説明する．4. で提案手法の性能評価を行い，最後に 5. で本稿のまとめと今後の課題について述べる．

2. 想定環境

2.1 Web サービス

各マイクロサービスは，複数人からなるチームによって開発が行われる．一般的には，設計から実装，運用まですべてを担当する 5,6 人からなるチームが理想とされている [6]．1 チーム

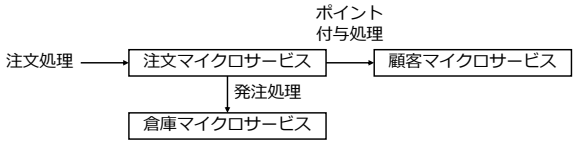

図 1　マイクロサービス・アーキテクチャの EC サイトにおける注文処理の例

が 1 マイクロサービスを担当する場合が多いが，1 チームで複数のマイクロサービスを開発・運用する場合もある．各マイクロサービスは異なる言語で実装されていてもよく，共通の規約に従った Web API さえ提供されていれば，他のチームが作成したマイクロサービスと疎結合に連携して動作できる．

マイクロサービス・アーキテクチャによって作成される Web サービスの例としては，オンラインショッピングサイトのような EC サイトが挙げられる．マイクロサービス・アーキテクチャの EC サイトは，例えば，顧客からの注文を処理する注文マイクロサービスや，商品の在庫や発送を管理する倉庫マイクロサービス，顧客の氏名や住所を管理する顧客マイクロサービスなど，複数のマイクロサービスによって構成される．新たに商品の購入者に対してポイントを付与するサービスを始める場合，顧客マイクロサービスを更新し，ポイント数を管理する機能を追加する．また，実際にポイントを付与する処理は，注文マイクロサービスから顧客マイクロサービスを呼び出すことによって実現できる．図 1 に，マイクロサービス・アーキテクチャの EC サイトにおける注文処理の例を示す．注文マイクロサービスは顧客からの注文を受けると，倉庫マイクロサービスに対して商品の発注処理を行うとともに，顧客マイクロサービスに対してポイント付与処理を実行する．

2.2　全体像把握の困難さ

2.1 で述べたように，マイクロサービス・アーキテクチャのシステムは複数のマイクロサービスが連携して動作する分散システムともいうことができ，システムの全体像を把握することは困難である．なぜなら，システムを構成するマイクロサービスの数は数十，多い場合で数百にもなり [2]，また，マイクロサービスは絶えず更新が行われ，新たなマイクロサービスも随時追加されるためである．しかし，障害や遅延の原因分析を行う際には，どのマイクロサービスがどのマイクロサービスを呼び出しているかという情報が必要になるため，システムの全体像を常に把握しておきたいというニーズがある．マイクロサービス間の呼び出し関係を可視化したものを，本稿ではフローと呼ぶ．

また，実際の Web サービスは，障害や負荷などへの対策のため，多重化されていることが多い．マイクロサービス・アーキテクチャの場合，処理が集中するマイクロサービスや，負荷が高いマイクロサービスなど，マイクロサービスの単位で多重化を行うことができる．あるマイクロサービスのインスタンス数は一定ではなく，時間や状況に応じてリアルタイムに変化していくことが考えられる．このような状況下では，事前に全体設計がわかっていたとしても，実際の通信経路を把握することは非常に難しくなる．そこで本稿では，実際の通信情報をもとに，リアルタイムのフローを可視化することで，障害や遅延の原因分析に役立てることを想定している．

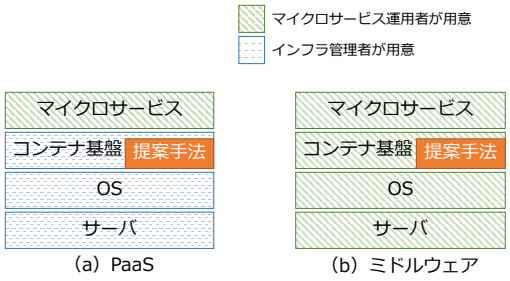

図 2　提供形態

2.3　提供形態

マイクロサービス・アーキテクチャの Web サービスが動作している環境として，次の 2 つのパターンを想定する．

2.3.1　PaaS

図 2(a) に示すように，例えば社内や大学内など，ある組織内で複数のマイクロサービスを稼働できるインフラがインフラ管理者によって用意されていると想定する．マイクロサービス運用者は，用意されたインフラの上でマイクロサービスを稼働させることで，Web サービスを運用する．この場合，インフラとしては Kubernetes [3] や Cloud Foundry [1] のようなコンテナ管理基盤が該当し，各マイクロサービスはコンテナアプリケーションとして実行されることが多い [5]．マイクロサービス運用者は，インフラ管理者と同じ組織に属している必要はなく，PaaS サービスとしてインフラ管理者が他の組織にインフラを貸し出しする場合も想定される．

インフラ管理者は，コンテナ管理基盤などが稼働しているサーバ上で通信ログを取得するか，通信ログを取得する仕組みを組み込んだコンテナ管理基盤を利用する．インフラ上で各マイクロサービス運用者が運用するマイクロサービスの通信を可視化するサービスをインフラの一部として提供することで，マイクロサービス運用者が障害分析などに役立てる．

2.3.2　ミドルウェア

図 2(b) に示すように，例えば社内や大学内など，ある組織内でマイクロサービス・アーキテクチャの Web サービスを運用する場合を想定する．マイクロサービス運用者は，マイクロサービスの稼働環境として，複数の物理サーバもしくは仮想サーバからなるインフラ環境を構築し，その上でマイクロサービスを稼働させることで，Web サービスを運用する．インフラとしては，前節と同じく Kubernetes や Cloud Foundry のようなコンテナ管理基盤が該当し，マイクロサービスはコンテナアプリケーションとして実行されることが多い．

マイクロサービス運用者は，マイクロサービスの通信を可視化するミドルウェアを追加でセットアップするか，あらかじめこの機能が組み込まれたコンテナ管理基盤などを利用することで，Web サービスの障害分析などに役立てる．

2.4　フローの利用方法

図 3 に示すように，マイクロサービス A がマイクロサービス B を呼び出し，マイクロサービス B がさらにマイクロサービス

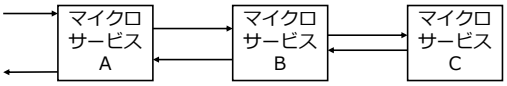

図 3　呼び出し関係の例

C を呼び出している，というフローがあらかじめわかっているものとする．また，この環境には，他にも多数のマイクロサービスが稼働しているものとする．マイクロサービス A を利用するユーザへの応答に遅延が生じていると報告があった際に，マイクロサービス運用者は可視化されたフローをもとに，遅延の原因はマイクロサービス A，B，C のいずれかにあると絞り込むことができる．このように，フローがあらかじめわかっていれば，遅延発生時の原因特定に役立てることができる．また，ある時間帯にフローを可視化したとき，マイクロサービス B からマイクロサービス C への呼び出しが現れなかった場合，マイクロサービス B か，マイクロサービス B からマイクロサービス C への通信経路になんらかの障害が発生したと推測することができる．

3. 提 案 手 法

本章では，各サーバで取得できる通信情報（送信元 IP アドレス，送信先 IP アドレス，送信先ポート番号，送信時刻，受信時刻）をもとにマイクロサービス間の呼び出し関係を推測し，可視化する手法を提案する．提案手法では，ある通信に対して，その通信（子通信）が発生するきっかけとなった通信（親通信）が存在すると仮定し，子通信の送信時刻からさかのぼって閾値以内の時間に受信した通信を親候補通信とする．この処理を一定量の通信ログに対しておこない，親候補通信の中から不要なものを削除し，残ったものを親通信と推測する．

3.1 概　　要

提案手法では，一定時間通信情報を取得し，その後，取得した通信情報をもとにフローを可視化する．通信情報として用いるのは，送信元 IP アドレス，送信先 IP アドレス，送信先ポート番号，送信時刻，受信時刻であり，通信の内容は取得しない．通信情報の取得方法としては，tcpdump コマンドを利用する方法や，マイクロサービスが稼働している基盤側で通信を行う際にこれらの情報をログとして出力する方法が考えられる．

提案手法では，ある通信が発生した場合には，必ずその通信が発生するきっかけとなった通信が存在すると仮定することで，通信同士の呼び出し関係を推測する．各マイクロサービスのインスタンスは固有の IP アドレスとポート番号の組 H をもつものとし，任意の H_i から H_j への通信が存在した場合，H_i をもつマイクロサービスから H_j をもつマイクロサービスへの Web API 呼び出しが発生したと考える．マイクロサービス・アーキテクチャのシステムではデータベースや他のサービスへの通信も想定されるが，あるホスト上の特定のポートで接続を待つという動作はマイクロサービスと共通の動作である．そのため，提案手法ではマイクロサービスとこれらを区別しなくてよい．

図 4 に示すように，提案手法では任意の H_i（図中では 172.17.96.6:9041）に着目し，その IP アドレスを送信元としてもつ通信（図中では a，b）に対し，送信時刻からさかのぼって閾値 TH の間に受信した通信すべて（図中では a に対する 1，2，b に対する 3，4）を，親候補通信として列挙する．閾値 TH の間に複数の IP アドレスからの通信を受信していた場合，どの IP アドレスからの通信が親通信なのかを確定できない．そのため，不要な親通信候補のみを除外し，残りの親通信候補はすべてフローに出力する．

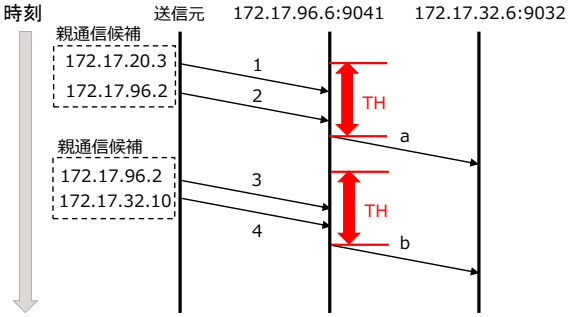

図 4　提案手法の概要

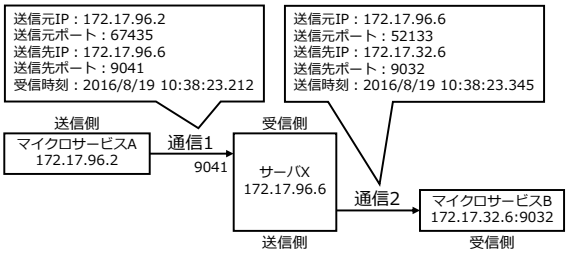

図 5　親子関係の例

3.2 通信の親子関係

図 5 に示すように，通信 1 が通信 2 を呼び出した状況を，サーバ X において観測した場合を考える．ここで，通信 1 と通信 2 は親子関係にあるといい，通信 1 は通信 2 の親通信，通信 2 は通信 1 の子通信と呼ぶ．このとき，サーバ X からは通信 1 の送信時刻と通信 2 の受信時刻は観測できない．

提案手法では，マイクロサービス間の呼び出し関係を推測する際，単一のサーバで取得した通信情報のみを利用する．これは，送信側と受信側両方の通信情報を関連付けて利用しようとすると，送信側で得られた通信情報と受信側で得られた通信情報を紐付けるために，通信自体に ID などの特定可能な情報を付加する必要があるためである．図 5 の例では，サーバ X において観測した情報のみを用いると，通信 1 の直後に通信 2 が発生したことは判別できるが，通信 2 が通信 1 によって呼び出された通信なのか，他の通信によって呼び出された通信なのか，もしくは通信 1 とは関連性がない通信なのかは判別できず，推測するしかない．

3.3 通信情報の取得

通信情報は，対象となる Web サービスに関連するマイクロサービスが稼働しているすべてのホスト上で取得する．例えば，マイクロサービスがコンテナ上で稼働している場合はそのホストとなっている物理マシンのすべてのポートを対象に，VM のような仮想マシンの上で稼働している場合はすべての仮想マシ

ンのすべてのポートを対象に tcpdump を実行することで，関連する通信情報を一括して取得することができる．また，通信が発生した際にコンテナ管理基盤側で通信情報をログとして出力する方法でもよい．

可視化のために利用する通信情報の量は，フローを可視化する目的によって異なる．例えば，常に通信ログを取得しておき，障害が発生した際は過去 1 時間分の通信ログを用いてフローを可視化して障害箇所の特定に役立てる方法や，過去 5 分間の通信ログをもとにフローを可視化し，Web サービスの状態監視に利用する方法などが考えられる．

通信ログに含まれない通信はフローには現れないため，できる限り長い時間の通信情報をもとにフローを可視化することも考えられるが，マイクロサービス・アーキテクチャのシステムでは頻繁に構成が変化するため，構成が変化する前後の通信ログは分割して分析することが望ましい．この場合，ある構成変更から次の構成変更までを一区切りとしてフローを可視化する方法が考えられる．ここでいう構成の変化とは，新たなマイクロサービスの追加や，負荷変動によるマイクロサービスのインスタンス数変更などを指す．

ここで，ホストが m 台あるとし，あるホスト $H_k(1 \leqq k \leqq m)$ で取得したすべての通信情報に含まれる通信を $C_k = \{c_{k,1}, c_{k,2}, \ldots, c_{k,n_k}\}$ とする．また，通信 $c_{k,i}$ の送信元 IP アドレスを $send_{k,i}$，送信先 IP アドレスを $recv_{k,i}$，発生時刻を $t_{k,i}$ とする．$t_{k,i}$ が送信時刻を表すか受信時刻を表すかは，$c_{k,i}$ が送信側で観測された通信か受信側で観測された通信かによって異なる．tcpdump などのツールで通信情報を取得すると，スリーウェイハンドシェイクのようなデータ送信をともなわない通信も取得されるため，これらの通信はあらかじめ除外しておき，C_k には含まれないものとする．また，$send_{k,i} = FS(c_{k,i})$ となるような $c_{k,i}$ から $send_{k,i}$ への写像 FS を定義する．

3.4 親通信候補の列挙

一定時間通信情報を取得したのち，親通信候補の列挙を行う．親通信候補の集合 $C_{k,i}$ は，次のように表される．

$$C_{k,i} = \{c_{k,j} \mid recv_{k,j} = send_{k,i}, t_{k,i} - TH < t_{k,j} < t_{k,i}\} \quad (1 \leqq j \leqq n_k) \quad (1)$$

tcpdump などでまとめてホスト上の全ポートの通信情報を取得した場合，他ホストから自ホスト宛の通信も取得されるが，そのような通信の親通信候補は自ホスト上では取得できないため，親通信候補は空集合となる．すなわち，$C_{k,i} \neq \emptyset$ の場合，$send_{k,i}$ はホスト H_k 上の仮想ホストもしくはコンテナなどの IP アドレスであるため，$t_{k,i}$ は送信時刻，$t_{k,j}$ は受信時刻となる．よって，$C_{k,i}$ は，$send_{k,i}$ の送信時刻からさかのぼって TH 以内に $send_{k,i}$ 宛に受信した通信すべての集合を表す．

3.5 親通信候補の集計

ある IP アドレス ip に対して，$send_{k,i} = ip$ となる i は複数存在する場合が多い．これは，通信情報を取得した期間内に，ip で要求を待ち受けているマイクロサービスに対する Web API

対象通信の送信元IPアドレス：10.254.131.37

ID	親通信候補の送信元IPアドレス	対象通信の送信先
1	172.17.41.8	172.17.94.2
2	172.17.41.8	172.17.94.2
3	172.17.94.8	172.17.94.2
4	172.17.41.8, 172.17.94.8	172.17.94.2

集計結果：10.254.131.37

親対象通信の送信先	親通信候補の送信元IPアドレス	出現回数
172.17.94.2	172.17.41.8	3
172.17.94.2	172.17.94.8	2

図 6　親通信候補の集計

対象通信の送信元IPアドレス：10.254.23.165

ID	親通信候補の送信元IPアドレス	対象通信の送信先
1	-	172.17.94.5
2	-	172.17.94.5
3	172.17.94.1	172.17.94.5
4	-	172.17.94.5

集計結果：10.254.23.165

親対象通信の送信先	親通信候補の送信元IPアドレス	出現回数
172.17.94.5	-	3
172.17.94.5	172.17.94.1	1

図 7　不要な親通信候補の除外

呼び出しが複数回存在することを意味している．親通信候補 $C_{k,i} = FC(send_{k,i})$ は，たとえ ip が同一であっても i が異なると，異なる集合になる．そこで，$send_{k,i} = ip$ となるすべての親通信候補 $C_{k,i}$ を集計することで，呼び出し元のマイクロサービスを特定する．

図 6 に，親候補の集計方法を示す．送信元 IP アドレスが 10.254.131.37 である通信が 4 回存在し，各通信の親通信候補の送信元 IP アドレスをそれぞれ示している．ここで，対象通信の送信先ごとに親通信候補の送信元 IP アドレスを数え上げることで，出現回数を集計する．

3.6 不要な親通信候補の除外

親通信候補 $C_{k,i} = FC(send_{k,i})$ が空集合になる場合，$send_{k,i}$ がホスト H_k 上の IP アドレスではないか，もしくは親通信がもともと存在しない通信であると考えられる．親通信が存在しない通信の例としては，例えば 1 時間に 1 回ログを送信する場合など，一定期間ごとに自動的に実行される処理が挙げられる．したがって，親候補通信が一度でも存在しない通信には親通信が存在しないはずと仮定し，親通信の推測はおこなわない．

図 7 に，親候補通信の集合が空集合になる場合の処理を示す．送信元 IP アドレスが 10.254.23.165 である通信が 4 回存在し，そのうちの 3 回は親候補通信の集合が空集合であることを示している．この場合，10.254.23.165 から 172.17.94.5 の通信には親通信が存在しないと仮定し，親通信の推測はおこなわない．

3.7 親通信の推測

ある通信に対する親候補通信が複数存在する場合，それらの送信元 IP アドレスを確認し，同一のものは 1 つにマージする．これは，あるマイクロサービスが非常に短い間隔で同じマイク

表 1　パラメータ

パラメータ名	値
TH	$1, 2.5, 5, 10, 25, 50, 100, 250, 500,$ $1000, 2500, 5000, 10000(ms)$
通信情報の取得時間	約 1 時間
取得した通信数	42,602

ロサービスから呼び出された場合，TH 内に同一の送信元 IP アドレスからの通信が複数回含まれるためである．

ある送信元 IP アドレスと送信先 IP アドレスの組をもつ通信の集合に対して，親候補通信の送信元 IP アドレスが 1 通りであれば，その IP アドレスを親通信の送信元 IP アドレスと推測する．しかし，親候補通信の送信元 IP アドレスが 2 通り以上存在する場合，誤って抽出された親候補通信が含まれている可能性が考えられる．一方で，例えばマイクロサービスが多重化されており，同一マイクロサービスのインスタンスが複数稼働している場合，それらは異なる IP アドレスをもつものの，いずれも親通信といえる場合もある．本稿では，親候補通信の送信元 IP アドレスが 2 通り以上存在する場合，いずれも親通信であると推測する．

4. 評　　価

本章では，提案手法の評価を行う．本評価では，提案手法において閾値 TH を変化させた際に得られるフローの確からしさについて評価する．

4.1 評 価 環 境

評価対象の Web サービスとして，我々の研究グループで日常的に利用しているチャットボット向けの情報収集システムを用いる．このシステムは 20 程度のマイクロサービスからなり，マイクロサービス間の通信は Web API を用いるものが大半であるが，一部 AMQP によるメッセージングを用いるものもある．また，MySQL や HAProxy といったサービスも含まれている．各マイクロサービスのインスタンス数は大半が 1 であるが，インスタンス数が 2 のマイクロサービスも存在する．

これらのマイクロサービスの目的は，主に社外や社内の Web サーバや Web API からデータを取得し，整形したのちにチャットボット経由でチャット画面に情報を表示することである．また，一部のマイクロサービスは GUI をもっており，Web ブラウザに情報を表示する．社内の通信環境による制限のため，マイクロサービスが社外へアクセスを行う際には，必ずプロキシサーバを経由する必要がある．マイクロサービスの大半にはタイムアウト時間が設定されており，最も長いタイムアウト時間は 10 秒，最も短いタイムアウト時間は 1 秒である．これらの Web サービス群へのアクセス経路は複数存在し，チャットコマンドによるチャットボット経由のアクセス，Web GUI 経由のアクセス，スケジューラや統合監視ソフトウェアである Zabbix [7] による定期的なアクセスなどである．

各マイクロサービスは Kubernetes によって管理されるコンテナ上で稼働している．各コンテナには Kubernetes によって固有の IP アドレスとポート番号が割り当てられる．また，Kubernetes には Service と呼ばれる機能が存在し，この機能によって，コンテナがいずれのホスト上にデプロイされていても一意の名前でアクセスが可能になるという，DNS のような役割を果たしている．

表 1 に，本評価で用いたパラメータを示す．通信情報はあらかじめ約 1 時間取得しておき，取得した通信数は 42,602 だった．この通信情報に対して表に示したように閾値 TH を変化させ，評価をおこなった．

4.2 評 価 結 果

図 8，図 9，図 10 に，それぞれ $TH = 1$，$TH = 1000$，$TH = 10000$ の場合に得られたフローを示す．フロー中の楕円形はマイクロサービス，楕円形の中の数字は IP アドレスもしくは IP アドレスとポート番号の組を表している．また，矢印はマイクロサービス同士の親子関係を表し，あるマイクロサービスの左側の矢印が親通信，右側の矢印が子通信を表している．矢印上の数字は，その通信の発生回数を表している．

図から，TH が大きいほど得られるフローが複雑かつ大きくなっていることがわかる．今回の評価で用いたマイクロサービス・アーキテクチャの Web サービスでは，常にすべての通信が発生しているわけではないため，正解のフローを知ることは難しい．そのため，正解率のような評価を行うことはできない．しかし，例えば図 8 において 10.25.212.100 から同じ 10.25.212.100:9035 への通信が現れている箇所や，図 9 において 10.25.212.100:3306 という MySQL から 172.17.94.3:9000 のマイクロサービスへの通信が現れている箇所など，いくつか誤っていると思われる部分が見受けられる．

また，図 10 の一番上部にあるフローは 4479 回となっているにもかかわらず，TH が小さい場合には現れていない．提案手法では，一度でも TH の時間内に親通信が含まれない場合は親通信候補から除外する，という処理を行うため，このように発生回数が多いにもかかわらず TH が小さい場合に除外されてしまうフローが存在する．本稿で対象としたマイクロサービスには最長 10 秒のタイムアウトが設定されており，TH が 10 秒以下の場合，なんらかのエラーによりタイムアウトが生じると，エラーが発生した通信はフローに現れなくなってしまう．

$TH = 10000$ では不要と思われる箇所も可視化されているものの，$TH = 1000$ では削除されてしまっている箇所があり，どちらのフローがよいと判断するのは難しい．いずれにしても，多少の誤りは見受けられるものの，それ以外の箇所は正しい親子関係がフローに現れており，マイクロサービス間の呼び出し関係を高い精度で推測できているといえる．

図 11 に，TH を変化させた際の子側の通信の内訳を示す．Pno は親通信候補が存在しなかった通信の数，Pdelete は親通信候補が除外された通信の数，Pexist は親候補が 1 つでも存在した通信の数を表している．

図から，TH が大きいほど親通信候補が存在する通信の数が増えていることがわかる．これは，TH が小さい場合，親通信が TH の範囲外に発生したことになってしまい，親通信候補が正しく抽出されなくなるためである．$TH \leqq 5$ の場合，Pexist が 0 となっており，親通信がまったく抽出されていない．一方

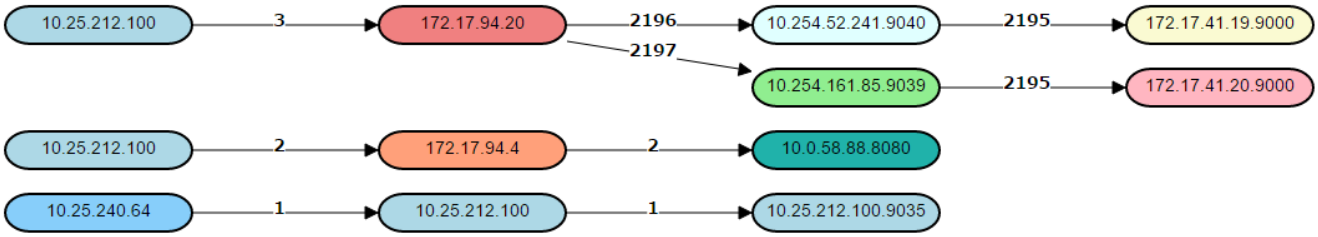

図 8　$TH = 1$ の場合に得られたフロー

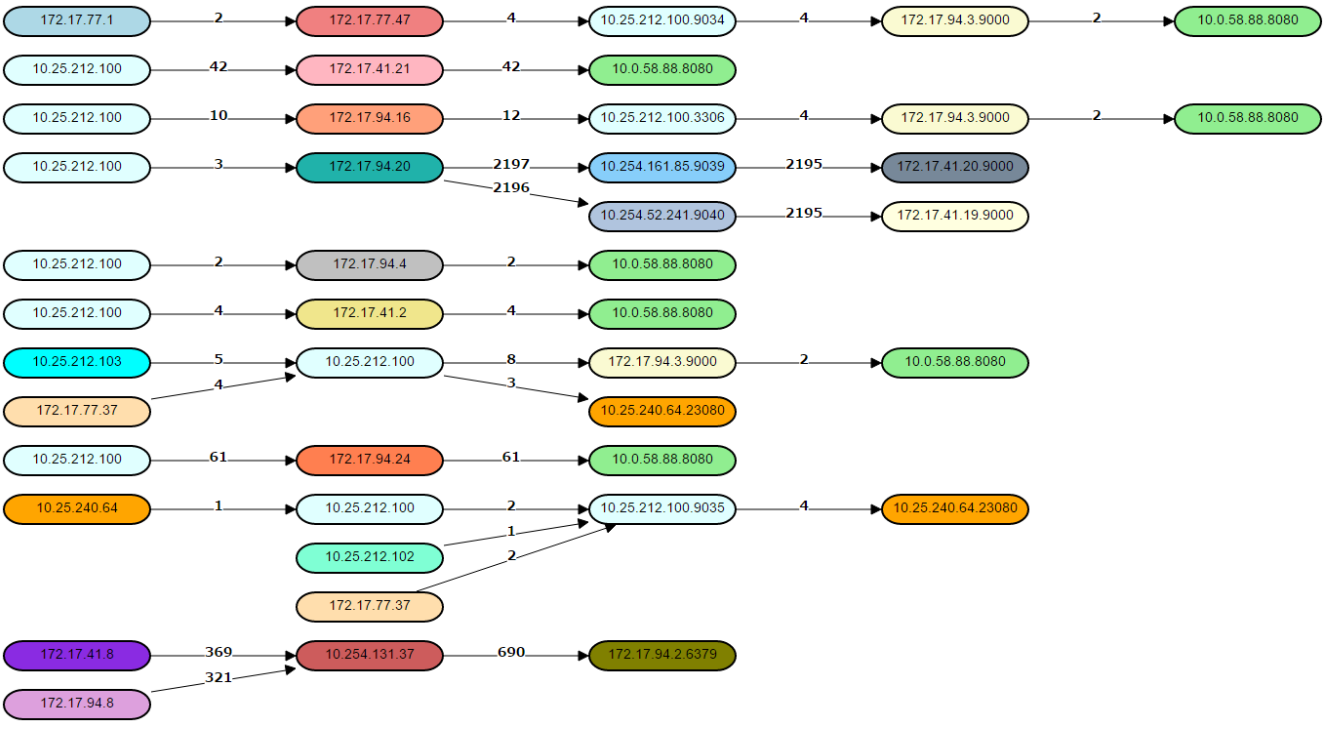

図 9　$TH = 1000$ の場合に得られたフロー

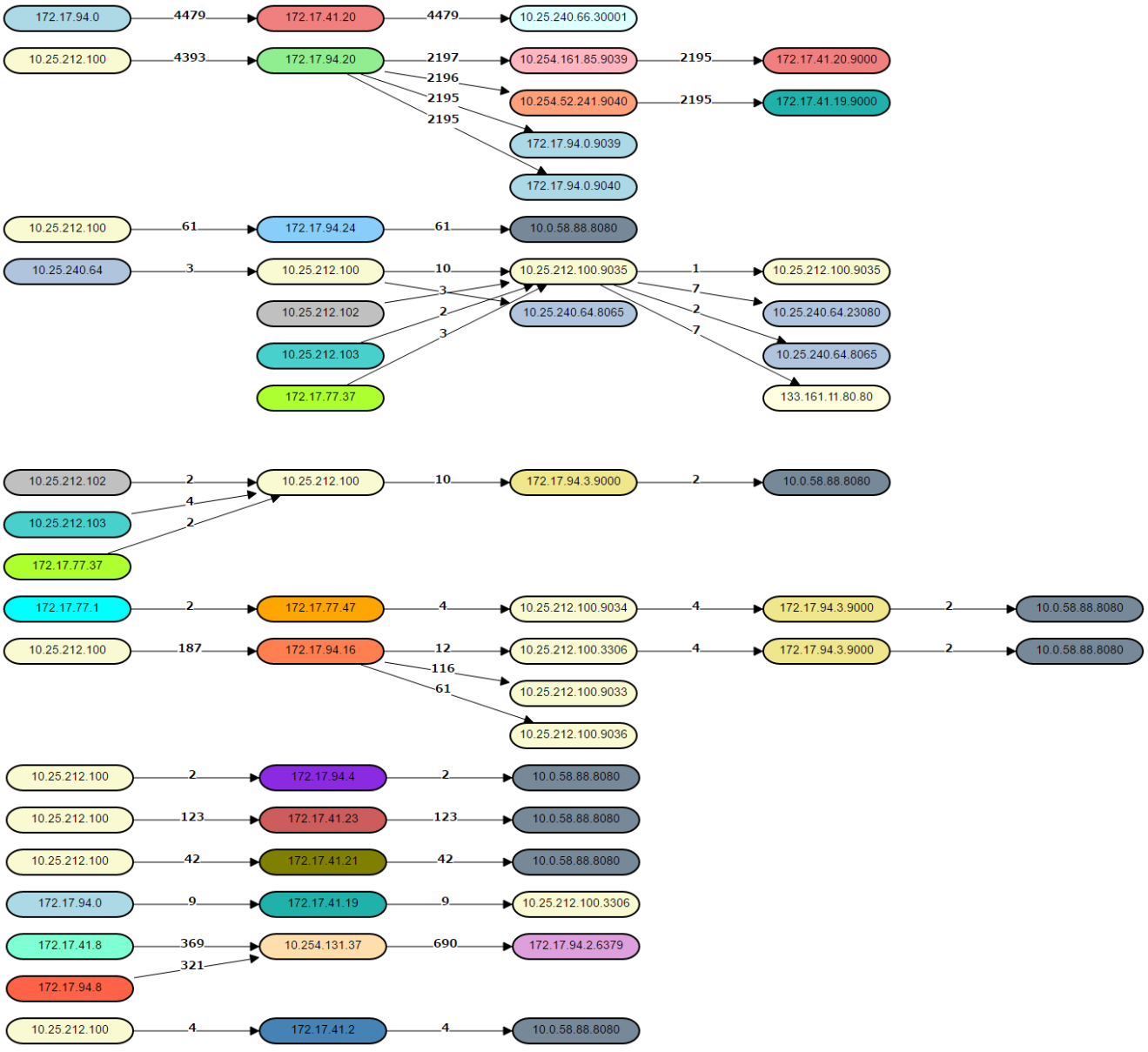

図 10　$TH = 10000$ の場合に得られたフロー

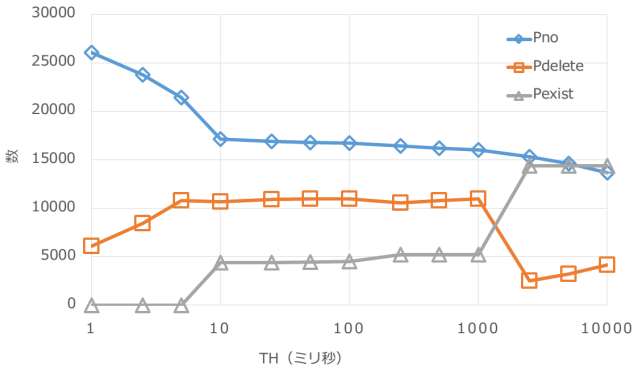

図 11　子側の通信の内訳

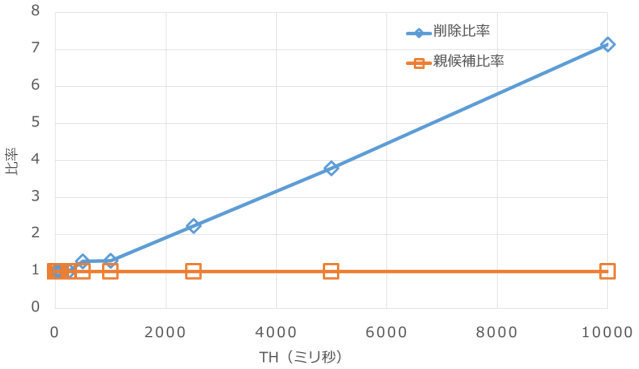

図 12　親通信候補が重複した割合

で，$TH \geqq 2500$ の場合，Pdelete は約 10 となっており，本来削除されるべきでない親通信候補が削除されなくなったために Pexist が増加したと考えられる．提案手法では，一度でも親通信候補が存在しない場合には親通信候補を削除することによって，親通信が存在しないはずの通信に対して親通信候補を挙げることを防いでいる．しかし，評価の結果から，この方法には短所があることがわかる．よって，例えば親通信候補が抽出されなかった回数が比較的少ない場合には，単純に親通信候補を削除してしまうのではなく，ある程度残すような手法を導入する必要があると考えられる．

図 12 に，TH を変化させた際の削除比率と親候補比率を示す．削除比率は 3.7 節におけるマージ後の親通信候補数をマージ前の親通信候補数で割った割合を表し，親候補比率はマージ後に残った親通信候補数を子通信数で割った割合を表す．

図から，TH が大きいほど削除割合が大きくなっていることがわかる．本稿の評価環境では，基本的に 1 つのマイクロサービスが 1 つの IP アドレスをもつため，親通信候補として抽出される通信は，ほとんどが同一のマイクロサービスからの Web API アクセスである．TH が大きくなると，本来の親通信と，それ以前に受信した親通信の複数が親通信候補として抽出され，結果として同一の送信元である親通信候補として削除されるため，削除比率が大きくなる．一方，親候補比率はほぼ 1 であり，複数の親通信候補から最適な親通信を選択すべき場面はほとんどなかったといえる．これも，本稿の評価環境では，基本的に 1 つのマイクロサービスが 1 つの IP アドレスをもつため，誤った親通信候補が現れにくいことによる結果である．

これらの評価結果から，本稿における評価では $TH = 1000$ が最適な TH ではないかと考えられる．これは，TH が 1000 より大きい場合，削除比率が大きく増加していることから，1 回以上前の親通信を親通信候補として挙げていると考えられるためである．しかし，最適な TH は各マイクロサービスの呼び出し頻度に応じて異なると考えられることから，環境に応じて最適な TH を設定する必要がある．

5. ま と め

本稿では，各サーバで取得できる通信情報（送信元 IP アドレス，送信先 IP アドレス，送信先ポート番号，送信時刻，受信時刻）をもとにマイクロサービス間の呼び出し関係を推測し，可視化する手法を提案した．実際に稼働しているマイクロサービス・アーキテクチャのシステムにおいて通信ログを取得し，評価を行った結果から，提案手法を用いることで，マイクロサービス間の呼び出し関係を高い精度で可視化できることを確認した．

今後の課題としては，以下の項目が挙げられる．

- 不要な親通信候補を削除する際に，必要な親通信候補まで削除しないように調整する方法を検討する．
- マイクロサービスごとに最適な TH を算出する方法を検討する．
- 複数の親通信候補が得られた場合に，その中から最適な親通信を決定する方法を検討する．
- 正解のフローがあらかじめわかるような評価環境を準備し，提案手法の正解率を評価する．

文　献

[1] “Cloud Foundry | The Industry Standard for Cloud Applications,” https://www.cloudfoundry.org/, 参照 Jan. 2017.
[2] D. Bryant, “Scaling Microservices at Gilt with Scala, Docker and AWS,” InfoQ, https://www.infoq.com/jp/news/2015/05/scaling-microservices-gilt, Apr. 2015.
[3] “Kubernetes - Production-Grade Container Orchestration,” http://kubernetes.io/, 参照 Jan. 2017.
[4] J. Lewis and M. Fowler, “Microservices,”, MARTINFOWLER.COM, http://martinfowler.com/articles/microservices.html, Mar. 2014.
[5] J. Stenberg, “Microservices, Containers and Docker,” InfoQ, https://www.infoq.com/news/2014/12/microservices-docker, Dec. 2014.
[6] S. Newman，佐藤直生（監修），木下哲也（翻訳），マイクロサービスアーキテクチャ，オライリージャパン，Feb. 2016.
[7] “Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution,” http://www.zabbix.com/, 参照 Jan. 2017.
[8] “OpenZipkin · A distributed tracing system,” http://zipkin.io/, 参照 Jan. 2017.
[9] B. H. Sigelman, L. A. Barroso, M. Burrows, P.=Stephenson, M. Plakal, D. Beaver, S. Jaspan, C. Shanbhag, “Dapper, a Large-Scale Distributed Systems Tracing Infrastructure,” Google Technical Report dapper-2010-1, Apr. 2010.