Vol.2021-IOT-52 No.16
2021/3/1

マイクロサービス型システムの監視における グラフDB採用によるメトリック問い合わせ最適化の検討

宮越 七菜[1,a)]　林 友佳[1]　中田 裕貴[1]　松原 克弥[1,b)]

概要：システムの拡張性や可用性などを向上する設計手法のひとつとして，マイクロサービス・アーキテクチャが注目されている．一方，マイクロサービス型システムでは，負荷に応じたスケールアウトなどの対応により，コンポーネントの数やそれらの依存関係が動的に変化する．分散システムの障害が複数要因の組み合わせで起きる傾向とあいまって，これら依存関係の動的変化は，マイクロサービス型システムの監視における障害発生時の状態把握や障害原因の追究を難しくしている．著者らは，依存関係の動的変化をともなう分散システムを対象とした監視において，コンポーネント間の依存関係に着目しつつ，システムの状態を迅速に把握できる可視化手法を提案している．本研究は，マイクロサービス型システムの監視において収集されるメトリックを管理するデータベースとして，グラフ DB の採用を検討する．本稿では，依存関係に着目したシステム状態可視化にともなって発行されるメトリック問い合わせを分析することで，依存関係情報や各メトリックの保存と探索におけるグラフ DB の有用性を議論する．また，監視ツールの評価に用いるためのマイクロサービス型システムの監視テストベッドの構築についても紹介する．

キーワード：システム監視，モニタリング，SRE，データベース，超分散システム

1. はじめに

マイクロサービス・アーキテクチャ [1] は，開発時の高い生産性や迅速なデプロイの実現，システムの拡張性や可用性向上といった目的で用いられることが増えている設計手法であり，実際，近年の多くのシステムで採用されている [2], [3], [4]．マイクロサービス・アーキテクチャにもとづいて設計されたシステム（以降，マイクロサービス型システム）は，物理サーバや VM ／コンテナといった複数の計算ノードへシステムを構成する各機能コンポーネントを分散配置して，ネットワークを介してそれらを疎結合することで動作する．このような分散システムのかたちでシステムを構築することにより，例えば，前述の可用性向上に関して，負荷に応じてコンポーネントの数を増やすスケールアウトや異常が起きたコンポーネントを予備のコンポーネントに切り替えるフェイルオーバーなどの仕組みを実現することが容易となる．

マイクロサービス・アーキテクチャにもとづいたシステム設計では，可用性向上などのメリットがある一方，実装されたシステムの運用監視における複雑性が増加するという課題がある．システム監視では，システムの入出力やシステムの状態や実行しているプラットフォームの状態を記録し続けることでシステムの状態を把握し，システムに障害が発生した際には，過去に遡って記録した状態変化を確認することで，障害の要因を特定する [5]．マイクロサービス型システムのような分散システムにおいて発生する障害には，複数の要因の組み合わせで発生する傾向がある [6]．そのため，分散システムにおける障害原因の特定において，コンポーネント間の依存関係の把握が特に重要となる．さらに，マイクロサービス型システムでは，依存関係が動的変化することで，たとえシステム設計者や開発者であっても，システム内の各コンポーネントの依存関係を常に把握することが困難となる．

著者らは，依存関係の動的変化をともなう分散システムを対象として，コンポーネント間の依存関係に着目したシステム状態把握に最適化した監視インタフェースを提案している [7]．本監視インタフェースは，Situation Awareness（状況認識）や認知心理学の観点から求めた要件にもとづいた可視化機能を実現している．コンポーネント間の依存関係を可視化することでシステム全体の状態を俯瞰的に把握することを支援する．さらに，障害が発生した際は，障害を検知したコンポーネントと障害発生時点から過去一定

[1] 公立はこだて未来大学
Future University Hakodate
[a)] b1017019@fun.ac.jp
[b)] matsu@fun.ac.jp

期間の間に依存関係のあったコンポーネントを迅速に特定し，それらコンポーネントの各メトリックを比較しながら確認できるようにすることで，平常時からの変化の把握とその起因となった箇所の特定を支援する．

本稿では，前述の提案インタフェースを実装した監視ツールへ採用すべきデータベース (以降，DB）について論じる．動的変化する依存関係にもとづいたシステム状態の可視化にともなって発行されるメトリック問い合わせの特性を分析するとともに，それら問い合わせに対するグラフ DB の有用性を議論する．また，コンポーネント間の依存関係に関する情報だけでなく，従来のシステム監視においても収集される時系列メトリックをグラフ DB へ格納するためのデータモデル設計を示す．さらに，提案する監視インタフェースの評価のために構築を進めているマイクロサービス型システムの監視テストベッドを紹介する．

2. 依存関係が動的変化するマイクロサービス型システムの監視

マイクロサービス型システムでは，動作するプラットフォームやネットワークの状態，システムへのアクセス負荷などの変化にともなって，コンポーネント間の依存関係が地理的・時間的に変化する．

地理的な変化とは，マイクロサービス型システムの構成が変化するさまを指す．例として，マイクロサービス型システムへのアクセスが増加し，サーバに負荷がかかった場合に新たなサーバが立ち上がるといったように，負荷に応じてコンポーネントの状態や構成，依存関係は動的に変化する [8]．時間的な状態変化とは，リクエスト数や CPU 使用率など，各コンポーネントに間する定量的な値の変化のことを指す．これらの状態変化が頻繁に起こることから，マイクロサービス型システムでは，障害が発生した際にシステムにどのような変化が起きたかを把握していないと，障害のきっかけがわからなくなるという課題がある．よって，リアルタイムな情報を素早く認識することで，システムに起きる地理的・時間的な状態変化を都度把握する必要がある．

依存関係が複雑であるというのは，地理的・時間的な状態変化を経て，コンポーネント間の依存関係にも変化が起こることが原因である．そのため，常にシステム全体の依存関係を把握することは難しい [9]．また，前章で述べたとおり，分散システムの障害は複数の要因の組み合わせで発生する傾向がある．その場合，監視ツールは要因ごとにアラートを発するため，システム全体としては大量のアラートが発生してしまう．これらのことから，マイクロサービス型システムでは，なにが障害の本当の要因かがわかりづらくなってしまうという課題がある．よって，障害の根本的な要因を依存関係をもとに追求することが必要である．

2.1 依存関係に着目する監視インタフェースの提案

前述のように，依存関係の動的変化をともなうマイクロサービス型システムの監視では，コンポーネント間の依存関係がシステムの状態変化に深く関与するため，依存関係に着目したシステム状態の把握が重要となる．この考えにもとづき，著者らは，動的変化する依存関係に着目することに最適化された新たな監視インタフェースをデザインした [7]．本監視インタフェースでは，地理的・時間的な状態変化を同時に表示することと，視覚的示唆による強調表現と俯瞰的表示の両立を行うことである．地理的な状態変化とは，コンポーネント間の依存関係の動的な変化や，コンポーネントの置かれる状況の変化を指す．具体的には，コンポーネントを円などの図形として描画し，ネットワーク通信関係にあるコンポーネント間に線を描画することで，依存関係の変化など地理的な状態変化を表している．時間的な状態変化とは，各コンポーネントに関する定量的な値の変化を指す．具体的には，コンポーネントを表す図形をクリックした際に描画されるウィンドウ内に，CPU 使用率やメモリ使用量などの時間ごとの値を表す折れ線グラフを描画することで，時間的な状態変化を表している．この特徴によって，コンポーネント間の依存関係やその変化，それぞれのコンポーネントの値の変化を迅速に把握できるようにしている．視覚的示唆による強調表現は，コンポーネントを表す図形の描画位置の調整や，色分けによって行っている．図形の描画位置の調整は，地理的変化や時間的変化が頻繁に起こるコンポーネントが，重点的に監視する必要のあるコンポーネントだと捉え，画面上部に描画するといったことである．色分けは，CPU 使用率やメモリ使用量の値が大きい場合はコンポーネントを表す図形の枠線の色を赤色にし，値が中くらいの場合は黄色，小さい場合は緑色とするといったことである．これらの強調表現と，多くのコンポーネントを同時に表示できる俯瞰的表示を同時に行うことで，ユーザにかかる認知的負荷を軽減しつつ，システムの状態変化を迅速に把握できるようにしている．

図 1 は，監視インタフェースの画面スナップショットである．この画面では，コンポーネントを表す図形と依存関係を表す線をメインで表示し，地理的な状態変化を可視化している．監視エンジニアは，この可視化されたグラフ構造からコンポーネント間の依存関係を把握しつつ，着目すべきコンポーネント集合を特定し，該当コンポーネント集合の CPU 使用率やメモリ使用量の時系列変化を比較・確認できる．表示されているいずれかのコンポーネントをユーザがクリックすると，そのコンポーネントの CPU 使用率やメモリ使用量を表す折れ線グラフを表示する．図 1 内の例では，front-end と order コンポーネントの CPU 使用率やメモリ使用量を表す折れ線グラフを表示している．特定のコンポーネントを選択してから，三角形のボタンをクリックすると，選択されたコンポーネントの依存関係を巻

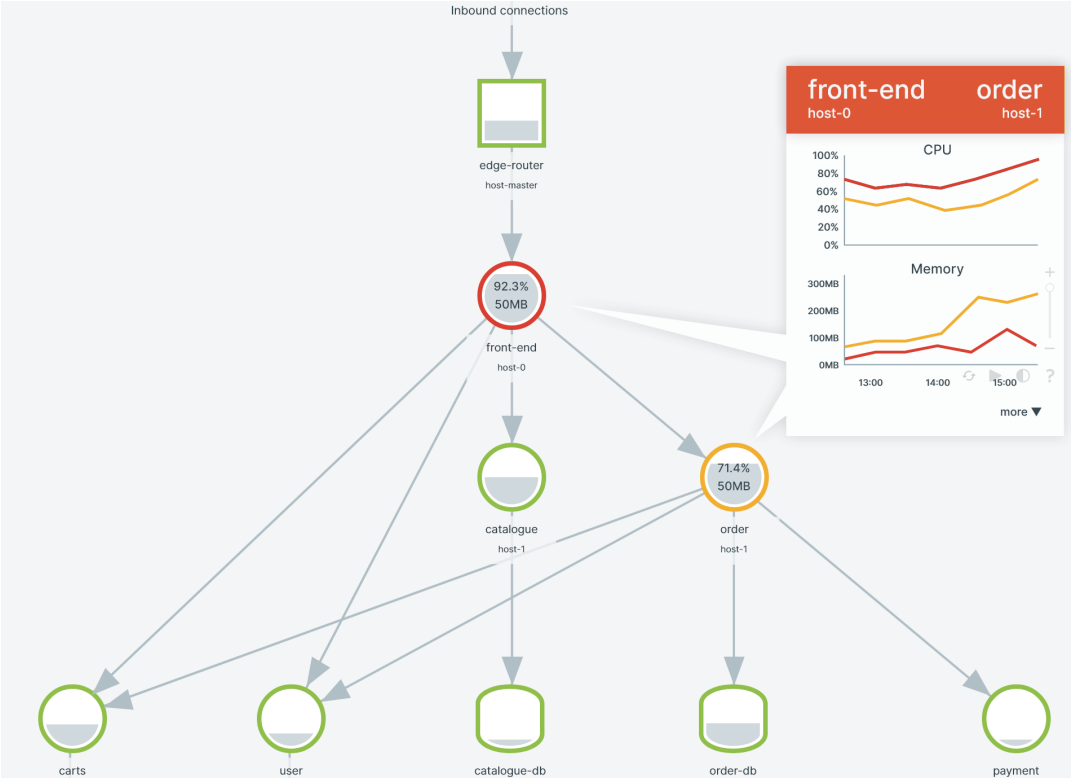

図 1　依存関係への着目に最適化された監視インタフェース

き戻して表示する．ユーザはこの機能を使うことで，障害発生後でも，障害発生時の依存関係の変化を確認できる．

3. 依存関係に着目する監視におけるメトリック問い合わせの特性

前章で述べた監視インタフェースの特徴やユースケースを踏まえると，必要な監視データは，コンポーネントのCPU 使用率やメモリ使用量などの監視メトリックと，コンポーネント間の依存関係である．監視データの取得タイミングは，最初に表示される画面を表示している間，コンポーネントをユーザがクリックしたとき，巻き戻し機能を使った時の 3 種類ある．最初に表示される画面を表示している間は，常に全てのコンポーネントの最新の監視メトリックと，全ての依存関係を取得する．監視インタフェース上でコンポーネントをクリックしたときは，クリックされたコンポーネントの，過去からクリックされた時点までの CPU 使用率やメモリ使用量を取得する．監視インタフェース上でコンポーネントを選択してから，依存関係の変化を巻き戻して表示するときは，クリックされたコンポーネントに接続されている依存関係と，その依存関係の先にあるコンポーネントの監視メトリックを，過去の時間にさかのぼって取得する．

提案した監視インタフェースの実現にともなう問い合わせの特性は以下の 3 点である．1 点目は，動的に変化する依存関係の把握を主目的に設計されている監視インタフェースであるため，頻繁に全てのコンポーネントの情報と依存関係を取得する特性である．頻繁に依存関係を取得するため，依存関係の問い合わせが非効率であった場合，依存関係の変化に監視インタフェースが追従できなくなる可能性がある．2 点目は，監視メトリックなどの時間的に変化する値は，コンポーネントがクリックされた時のみ表示するため，時系列データの問い合わせ回数は，依存関係の問い合わせ回数より少ないという特性である．そのため，時系列データよりも依存関係の問い合わせの容易さを重視する必要がある．3 点目は，特定のコンポーネントから依存関係のあるコンポーネントをたどり，そのコンポーネントの情報を取得するという特性である．依存関係を辿ることが考慮されていない DB を用いて監視データを格納した場合，依存関係を辿るために詳細なデータの列挙や複雑なテーブルやデータの結合を行う必要がある．また，頻繁に依存関係が変化する環境では，これらの処理が大量に発生し，依存関係のあるコンポーネントの情報取得におけるボトルネックになる可能性がある．これらの特性から，動的に依存関係が変化する環境を対象とした監視インタフェースの実現には，依存関係の問い合わせを効率的に行える DB を用いて依存関係やコンポーネントの情報を格納することが必要である．また，効率的な問い合わせを実現するためには，時刻，依存関係，監視メトリックの 3 点を効率的に関連付けて格納，取得できることが有効である．

図 2　Neo4j の基本的なデータ構造

4. グラフ DB によるメトリック問い合わせ最適化

本稿では，動的変化する依存関係に着目した監視インタフェースを実装した監視ツールへ採用すべき DB としてグラフ DB[10] を採用することで，可視化にともなって発行されるメトリック問い合わせを最適化する．

グラフ DB は，関係性をもつデータモデルに特化している DB であり，関係の範囲指定や関係をたどる問い合わせが可能である．グラフ DB の例として，Neo4j[11] を挙げる．Neo4j の基本的なデータ要素を図 2 に示す．Neo4j はノード，リレーションシップ，プロパティというデータ要素を持つ．Neo4j 内に保存する個々の要素をノードとし，ノード間にある関係性をリレーションシップとして保存する．ノードとリレーションシップに関連する情報をプロパティとして key/value 形式で保存する．

グラフ DB を採用する理由は，依存関係をたどったメトリック問い合わせができるためである．グラフ DB 採用により，複数の DB に問い合わせる必要があった既存手法 [12] に対し，1 つの DB への問い合わせのみで監視データを扱うことが可能となる．グラフ DB は関係性を保持するために設計された DB であり，コンポーネント間の依存関係を保存する DB として適している．これにより，グラフ DB に依存関係と監視メトリック，取得時刻の 3 点を格納することで，着目しているコンポーネントから依存関係をたどり，たどった先のコンポーネントの監視メトリックを問い合わせるといった一連の分析がグラフ DB のみで完結する．また，グラフ DB は関係性をたどることによるデータの一括的な問い合わせができる．例として，把握しきれない依存関係をたどって障害原因を特定したいとき，リレーショナル DB への問い合わせは，コンポーネントの依存先のコンポーネントを何度も繰り返し指定するといった煩雑なクエリ記述が必要になり，問い合わせが非効率的になる．加えて，取得するデータを細かく列挙し，データの結合形式を厳密に記述する必要があるため，把握していない依存関係のなかから障害原因を探すことは難しい．一方，グラフ DB へ問い合わせる場合，依存関係をたどる回数を指定するのみでデータを一括抽出できるため，未知の依存関係のなかから障害原因を追究するための問い合わせに適している．

4.1 グラフ DB に監視データを格納する要件

可視化にともなう問い合わせで求められる監視データは 3 章で述べたとおり，コンポーネントごとの監視メトリック（CPU 使用率およびメモリ使用量）と動的変化する依存関係である．これらの監視データをグラフ DB に格納する要件として，依存関係や時系列データ，それらの取得時刻を関連付けて格納するデータ構造の定義と，データ操作方法の定義がある．

グラフ DB は関係性をもつデータモデルに特化しているため，ノード間に付与するリレーションシップを用いて依存関係を保存する．動的変化する依存関係が時刻ごとに変化しているとみなし，依存関係の取得間隔を監視メトリックの取得間隔と合わせ，同じタイムスタンプを持つノード間に依存関係を付与する．CPU 使用率やメモリ使用量といったコンポーネントの監視メトリックは，時刻ごとに計測した値を持つ時系列データである．本提案手法では，時系列データをグラフ DB に格納するため，コンポーネントの監視メトリックとタイムスタンプ（時系列データを取得した時刻）を一定時間ごとに取得し，新しいメトリックを取得するたびに新しいノードに記録する．ノード 1 つに保存するプロパティとして，コンポーネント名，タイムスタンプ，監視メトリック (CPU 使用率，メモリ使用量) を保存する．この保存構造により，ノードに保存された時系列値が計測されたコンポーネントと時刻を判別する．

データ操作方法は，依存の深さを指定した問い合わせを活用する．グラフ DB ではデータの範囲を指定した問い合わせが可能である．そのため，あるコンポーネントと依存関係を持ったコンポーネントの監視メトリックを問い合わせるとき，すべてのコンポーネントのノードを指定するより，依存の深さを指定するほうがグラフ DB に適している．依存の深さとは，あるノードから他のノードに辿り着くまでに経由するリレーションシップの数を指す．本提案手法では，3 章で述べた問い合わせを実現するため，時刻と依存の深さに着目したクエリを記述することでデータ操作方法を定義する．期間を指定する場合は，着目時刻と，着目時刻からさかのぼって問い合わせたい期間を用いて，タイムスタンプの範囲を指定するクエリを記述する．依存関係を持ったコンポーネントを問い合わせる場合は，依存の深さを指定したクエリを記述する．

5. グラフ DB を用いたメトリック問い合わせ評価環境の構築

マイクロサービス型システムの監視におけるグラフ DB を用いたメトリック問い合わせの評価を行うためには，監視対象となる動的に依存関係が変化する分散システム環境と分散システムを監視するテストベッド環境の構築，テストベッド環境から取得した監視データのグラフ DB への格納が必要である．

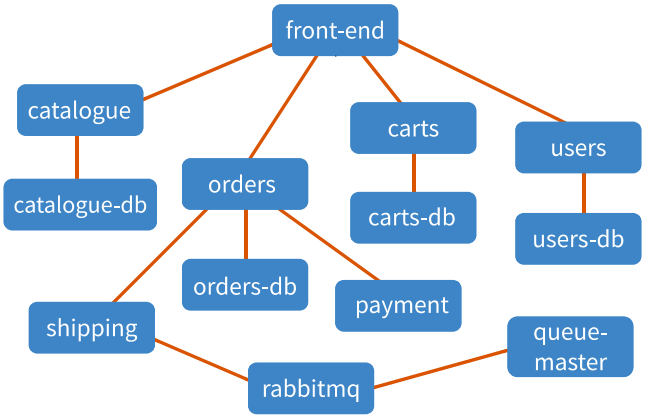

図 3　Sock Shop のサービス構成図

5.1　動的に依存関係が変化する分散システム環境の構築

マイクロサービス型システムをオーケストレーションシステムである Kubernetes[13] 上で動作させることで，動的に依存関係が変化する分散システム環境を構築した．Kubernetes 上で実行する監視対象システムとして，EC サイトを模したマイクロサービス型システムのサンプルである Sock Shop[14] を使用した．Sock Shop は，支払いを実現する payment や注文を実現する orders など，13 個のコンポーネントから構成される（図 3）．Sock Shop はサービスを構成するコンポーネントが多く，サービスの負荷状況に応じてコンポーネントの依存関係が複雑に変化するため，本研究の対象である動的に依存関係が変化する分散システムとして適している．

Sock Shop で依存関係の変化を適切に発生させるためには，サービスの負荷状況を定期的に変化させる必要がある．しかしながら，Sock Shop の特定のページにアクセスするだけでは，Sock Shop を構成する一部のサービス間でのみ通信が行われるため，依存関係の変化が起こりにくい．そこで，より依存関係が変化するよう，実際にユーザが Sock Shop を利用する場面を想定したシナリオに沿って定期的にアクセスを実現するストレスツールを開発した．このストレスツールでは，会員登録をする，商品をカートに入れる，住所を登録する，商品の注文をする，といった一連のアクセスで負荷を発生させる．

5.2　分散システムを監視するテストベッド環境の構築と監視データのグラフ DB への格納

Sock Shop のメトリック問い合わせをグラフ DB である Neo4j を用いて実現するには，Sock Shop のメトリックとコンポーネント間の依存関係の 2 種類の監視データをグラフ DB に格納する必要がある．メトリックとコンポーネント間の依存関係の 2 種類の監視データを取得可能な監視テストベッド環境を実現するために，Sock Shop を実行している Kubernetes クラスタ上で cAdvisor[15] と Prometheus[16]，Grafana[17]，Weave Scope[18] の 4 つの監視ツールを導入した．

cAdvisor は，Prometheus でコンテナ毎のメトリックを取得するために使用するコンテナリソース情報収集ツールである．Prometheus は，クラウド環境や Kubernetes 環境上で用いられる，サーバのリソース使用情報やソフトウェアの統計情報といったメトリックを収集し，時系列 DB に格納する監視ツールである．Grafana は，Prometheus で収集したメトリックを可視化し，システム監視に活用するための可視化ツールである．Weave Scope は，マイクロサービスのコンポーネント間の依存関係を取得できる監視ツールである．

メトリックの取得は，cAdvisor を用いてコンテナごとに収集した Sock Shop のメトリックを Prometheus で取得することで実現した．しかし，Prometheus で取得した Sock Shop のメトリックを Neo4j に格納するには，メトリックを Neo4j で認識可能な CSV 形式に変換する必要がある．本稿では，可視化ツールである Grafana の CSV 出力機能を用いて，Prometheus で取得したメトリックを CSV 形式に変換した．また，Grafana を経由して出力した CSV 形式のメトリックを Neo4j で認識するために，出力した CSV ファイルの一行目にデータ名を追記する整形作業を行った．

依存関係の取得は，依存関係を取得できる監視ツールである Weave Scope を用いることで実現した．Weave Scope の Web API を用いることで，JSON 形式のファイルとして依存関係などの監視データを取得できる．しかし，この監視データの中には，依存関係以外の不要な情報が多く含まれている．本稿では，3000 行ほどある JSON ファイルから，10 行ほどの依存関係の情報のみを抽出して整形してから CSV 形式に変換した．

5.3　監視データ格納構造とクエリ記述例

格納するメトリックの間隔は，メトリックを取得した間隔に合わせて 15 秒とする．15 秒ごとに計測するメトリックを格納するため，ノードごとにメトリックを保存する．時刻ごとにデータを保存したノードを図 4 に示す．ノードのプロパティとして，name，time，cpu，memory を保存する．これら 4 つが，key/value 形式でデータを保存するプロパティの key にあたる．name は，分散システムを構成するサーバやコンテナといった，コンポーネントの名前である．time はタイムスタンプを意味し，cpu と memory はそれぞれ cpu 使用率とメモリ使用量を保存する．図 5 に示す矢印はコンポーネント間の依存関係を表し，DEPENDENCE リレーションシップとする．同じタイムスタンプをプロパティに持つノード同士でのみ DEPENDENCE リレーションシップを持つように構築することで，時刻ごとに依存関係を保存する．

3 章で述べた可視化にともなう問い合わせを実現するため，4 章で述べた監視データ操作方法の要件にもとづき，

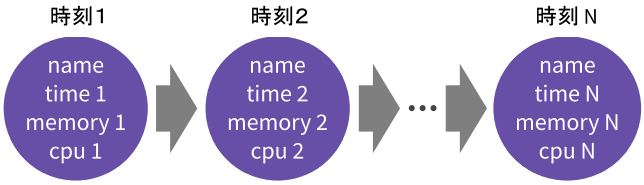

図 4　メトリックを保存するノード

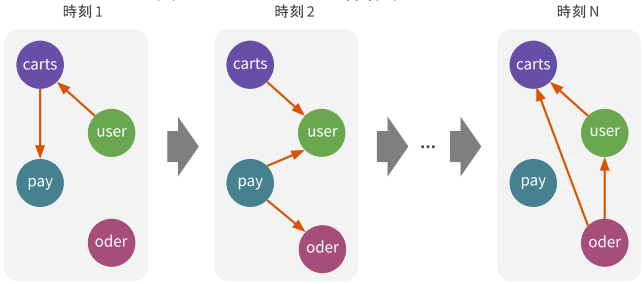

図 5　時刻ごとに保存する依存関係

クエリ記述例を挙げる．

全てのコンポーネントの最新のメトリックと，その依存関係を取得する問い合わせに用いるクエリとして，特定時刻のすべてのコンポーネントを問い合わせるクエリを図 6 に示す．2021 年 1 月 10 日 8 時 39 分 38 秒をタイムスタンプに持つノードを抽出し，特定時刻のすべてのコンポーネントのメトリックを問い合わせている．

特定のコンポーネントのメトリックの時間遷移を可視化するための問い合わせで必要となるクエリとして，指定期間内の特定コンポーネントのメトリックを取得するクエリを図 7 に示す．例として，時刻 2021 年 1 月 10 日 8 時 39 分 38 秒から過去 6 時間までの carts コンポーネントのメトリックを問い合わせたいとする．WITH 句では着目時刻の 2021 年 1 月 10 日 8 時 39 分 38 秒を変数 t に代入している．OPTIONAL MATCH 句を用いることで，リレーションシップを持たないノードも検索対象とする．ノードのプロパティとして name:”carts” を持つノードを検索している．’PT6H’ は 6 時間を指し，ノードを検索する条件として WHERE 句では，抽出したいノードのプロパティ time の期間を時刻 t から過去 6 時間前までに指定している．

依存関係の変化を巻き戻して表示するときに用いるクエリとして，指定期間の間，コンポーネントに接続されている依存関係と，その依存関係の先にあるコンポーネントのメトリックを取得するクエリを図 8 に示す．例として，2021 年 1 月 10 日 8 時 39 分 38 秒から 1 時間前までの期間内に，carts コンポーネントから依存の深さが 2 まで先のコンポーネントを問い合わせたいとする．5 行目までは前述と同様に，着目時刻 2021 年 1 月 10 日 8 時 39 分 38 秒から 1 時間前までの carts コンポーネントのノードを抽出し，変数 c に代入している．6 行目の MATCH 句では，変数 c と DEPENDENCE の深さが 1 から 2 まであるコンポーネントのノードと，それらの依存関係を抽出している．

```
1  MATCH (c:Component
2      {time:"2021-01-10T08:39:38.000Z"})
3  RETURN c
```

図 6　特定時刻のメトリックを問い合わせるクエリ

```
1  WITH "2021-01-10T08:39:38.000Z" AS t
2  OPTIONAL MATCH
3  (c:Component{name:"carts"})
4  WHERE DateTime(c.time) <= DateTime(t)
5  AND DateTime(c.time)
6      > DateTime(t) -duration('PT6H')
7  RETURN c
```

図 7　特定コンポーネントのメトリックを問い合わせるクエリ

```
1  WITH "2021-01-10T08:39:38.000Z" AS t
2  OPTIONAL MATCH
3  (c:Component {name:"carts",time:t})
4  WHERE DateTime(c.time) <= DateTime(t)
5  AND DateTime(c.time)
6      > DateTime(t) -duration('PT1H')
7  WITH c
8  MATCH
9  dep = (c)-[:DEPENDENCE*1..2]-(:Component)
10 RETURN dep
```

図 8　期間と依存関係の範囲指定したクエリ

6. 関連研究

マイクロサービス型システムを対象とした構成差分可視化ツールの研究 [12] では，リレーショナル DB である MySQL と，グラフ DB である Neo4j を組み合わせて，障害が発生した前後の構成変化を可視化することで，迅速に差分を把握することを可能にしている．本関連研究では，コンポーネントの追加・削除や依存関係の変更といった構成情報の差分はリレーショナル DB へ保存し，可視化に必要な情報のみを保存・取得するためにグラフ DB を用いている．マイクロサービス型システムを対象としている点や依存関係の時系列データを保存している点で本研究と関連が深い．しかし，本研究では，時系列データや依存関係情報のすべてをグラフ DB へ格納している点が異なる．

ネットワークセキュリティ状況認識（以降，NSSA）モデルを構築する研究 [19] では，異なる種別の DB を複数組み合わせる従来の NSSA モデル構築手法と比較して，グラフ DB を単一で用いた構築手法のほうが，問い合わせ処理の効率性やスケーラビリティの点で優れていることを示している．複数の DB を組み合わせる手法では，データの問い合わせや分析が非効率的になることを明らかにしている．対象データであるネットワークトポロジがグラフ構造であることに着目してグラフ DB を採用し，さらに，トポロジ

以外の性質の異なるデータもグラフ DB へ格納することで，複数 DB 間の連携オーバヘッドを削減し，より効率的な問い合わせ処理を実現している．本研究においても，グラフ構造をもつ依存関係の情報だけでなく，メトリックなどの性質の異なるデータも含めてグラフ DB へ保存している点が類似している．本研究で扱うコンポーネント間の依存関係やメトリックは，本関連研究が対象としているデータとは異なる．しかし，本関連研究からの知見により，グラフ DB のみの採用は，実現する監視ツールが問い合わせ処理性能やスケーラビリティにおいて有用であると推測する．

Transtracer [9] は， Linux カーネル内の各接続の終端点であるソケットに含まれる接続情報を監視することで，コンポーネント間の依存関係の変化を自動追跡することを可能にしている．依存関係に着目する監視ツールにおいて，マイクロサービス型システムで動的変化する依存関係の抽出に本研究成果が活用できると考える．

7. おわりに

本稿では，マイクロサービス型システムを対象とした監視ツールにおいて，依存関係情報や各メトリックの格納や探索の効率性に対して，グラフ DB の有用性を検討した．依存関係に着目した監視インタフェースへのインタラクションによって発行されるメトリック問い合わせを分析し，着目するコンポーネントから依存関係がある範囲のコンポーネント群を指定したメトリック問い合わせが重要となることを明らかにした．その結果，依存関係情報の格納や探索に特化したグラフ DB が有用であることを導き出した．一方，グラフ DB は，監視にともなって収集されるメトリックの特性である時系列データの格納に特化していないため，本検討において，グラフ DB へ時系列データを格納するためのデータモデルを定義した．さらに，本検討結果にもとづいて実現する監視ツールの評価を進めるため，オーケストレーションシステム Kubernetes とサンプル EC サイト実装 Sock Shop を用いたマイクロサービス型システムの監視テストベッドの構築について述べた．

今後，本検討結果にもとづいて，依存関係への着目に特化した監視インタフェースとグラフ DB を統合した監視ツールを実装する．本稿で検討に用いたプロトタイプ実装では，メトリック取得フレームワーク，グラフ DB，監視インタフェースが統合されておらず，連携に人的作業が必要となっている．監視ツールを実現するためには，監視メトリックを自動で取得し，グラフ DB に自動インポートする機構を実現する必要がある．また，本監視ツール内で発行されるメトリック問い合わせに対するグラフ DB の有用性を評価する．既存監視ツールの多くで採用されている時系列 DB を比較対象として，問い合わせの処理性能や問い合わせ記述の効率性などの観点で評価を行いたい．

参考文献

[1] Pahl, C. and Jamshidi, P.: Microservices: A Systematic Mapping Study, *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, p. 137146 (2016).
[2] M. Fulton Ⅲ, S.: What Led Amazon to its Own Microservices Architecture, https://thenewstack.io/led-amazon-microservices-architecture (2015 (accessed January 27, 2021)).
[3] Mauro, T.: Adopting Microservices at Netflix: Lessons for Architectural Design, https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices (2015 (accessed January 27, 2021)).
[4] Varshney, V.: Microservice Architecture Case Study on UBER's Microservice Architecture, https://medium.com/@vanshvarshney_/microservice-architecture-case-study-on-ubers-microservice-architecture-6380193ad551 (2020 (accessed January 27, 2021)).
[5] Poirier, G.: Monitoring is Dead (2016 (accessed January 30, 2020)). `https://speakerdeck.com/grepory/monitoring-is-dead?slide=48`.
[6] Yuan, D., Luo, Y., Zhuang, X., Rodrigues, G. R., Zhao, X., Zhang, Y., Jain, P. U. and Stumm, M.: Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems, *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 249–265 (2014).
[7] 林 友佳，伊勢田 蓮，松原 克弥，鷲北 賢，坪内 佑樹，松本亮介：動的適応性を持つ分散システムを対象としたシステム状態可視化手法の検討，技術報告 22 (2020).
[8] Matsumoto, R., Kondo, U. and Kuribayashi, K.: FastContainer: A Homeostatic System Architecture High-Speed Adapting Execution Environment Changes, *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, IEEE Computer Society, pp. 270–275 (2019).
[9] 坪内 佑樹，古川 雅大，松本 亮介：Transtracer: 分散システムにおける TCP/UDP 通信の終端点の監視によるプロセス間依存関係の自動追跡，インターネットと運用技術シンポジウム論文集，Vol. 2019, pp. 64–71 (2019).
[10] Robinson, I., Webber, J. and Eifrem, E.: グラフデータベース——Neo4j によるグラフデータモデルとグラフデータベース入門，オライリー・ジャパン (2015). 佐藤直生，木下哲也 訳.
[11] Neo4j: How to visualize time-based graphs with Neo4j, `https://neo4j.com/blog/visualize-time-based-graphs-neo4j/` (accessed August 1, 2020).
[12] 近藤 玲子，麻岡 正洋，渡辺幸洋ほか：マイクロサービス運用支援のためのコンテナからインフラまでの構成差分検出技術の開発，研究報告コンピュータセキュリティ (CSEC)， Vol. 2020, No. 2, pp. 1–6 (2020).
[13] Google: Kubernetes, `https://kubernetes.io/` (accessed January 26, 2021).
[14] WEAVEWORKS: Microservices Demo: Sock Shop, `https://microservices-demo.github.io/` (accessed October 28, 2020).
[15] Google: Analyzes resource usage and performance characteristics of running containers. - google/cadvisor, `https://github.com/google/cadvisor` (accessed January 24, 2021).
[16] Prometheus: Prometheus - Monitoring system & time se-

ries database, https://prometheus.io/ (accessed January 26, 2021).
[17] Labs, G.: Grafana: The open observability platform — Grafana Labs, https://grafana.com/ (accessed January 30, 2020).
[18] WEAVEWORKS: Weave Scope: Automatically Detect and Process Containers And Hosts, https://www.weave.works/oss/scope/ (accessed January 30, 2020).
[19] Tao, X., Liu, Y., Zhao, F., Yang, C. and Wang, Y.: Graph database-based network security situation awareness data storage method, *EURASIP Journal on Wireless Communications and Networking*, Vol. 2018, No. 1, p. 294 (2018).