「マルチメディア，分散，協調とモバイル
(DICOMO2019)シンポジウム」 令和元年7月

超個体型データセンターを目指した ネットワークサービス間依存関係の自動追跡の構想

坪内 佑樹[†1,a)] 古川 雅大[†2] 松本 亮介[†1]

概要：インターネットの利用者による多様な要求に応えるために，インターネットサービスを支えるシステムが大規模化かつ複雑化している．利用者とデータセンター間のネットワークの中間層を利用するエッジコンピューティングや小・中規模データセンターが地理的に分散する超個体型データセンターに向けて，今後はこれまで以上にシステムの規模と複雑性が高まっていく．その結果，システム管理者はシステムを構成するネットワークサービス同士の依存関係を把握することが難しくなるため，依存を自動で発見することが必要となる．しかし，依存を自動発見する先行手法は，システム管理者が把握できていない未知のネットワークサービスとの通信に対して，受動的にトラヒックを観測し，相関する活動を示すサービスを発見する．そのため，依存検出の偽陽性率が高いまたは依存関係の方向が識別できないという課題がある．そこで，本研究では，ネットワーク通信を終端する各ホスト上で確立されたトランスポート接続情報を網羅的に収集することにより，ネットワークサービス間の依存関係を自動で追跡可能な基盤を提案する．本提案により，未知のネットワークサービスであっても，OS がサポートするトランスポート接続を利用する限りは，依存を発見可能となる．また，接続を終端するホスト上にて，実際に発生したサービス間の接続を検出するために，偽陽性を削減できる．さらに，トランスポート接続の両端のプロセスを接続を要求する側と接続を待ち受ける側に分けることにより，前者が後者に依存すると判定できる．

A Concept of Automatic Tracing for Network Service Dependencies Toward Super Organic Data Center

Yuuki Tsubouchi[†1,a)] Masahiro Furukawa[†2] Ryosuke Matsumoto[†1]

1. はじめに

インターネットが人々の生活に欠かせないものとなっていることから，企業が 10 年以上の長期間に渡り，インターネットを利用したサービス（インターネットサービス）を，機能を追加しつつ，提供し続けるようになってきている．また，インターネットサービスが事業として成長するにつれて，利用者からのアクセス数が増大している．さらに，インターネットサービスの利用者による多様な要求に応えるために，単一の企業が，ソーシャルネットワーク，電子商取引および音声・動画配信など，多様なサービスを提供するようになってきている．あるインターネットサービスがもつ一部機能を他のサービスから利用する場合，互いにネットワーク接続することにより，機能を共通して利用することがある．このような長年の機能追加，利用者からのアクセス増加および複数のサービスの接続などの要因により，企業が管理するネットワークが大規模化かつ複雑化している．

インターネットサービスを提供するために，現在クラウドコンピューティングが広く利用されている一方で，IoT(Internet of Things)[1]，スマートシティ，仮想現実などのリアルタイム性が要求される領域において，利用者近傍のホームゲートウェイやネットワークルーター，小規模データセンター [2] といったようなネットワーク端（エッジ）に存在する資源を活用するエッジコンピューティング [3], [4], [5] が研究されている．加えて，松本らの研究 [6] では，小・中規模データセンター，あるいはデータセンター

[†1] さくらインターネット株式会社 さくらインターネット研究所
SAKURA Research Center, SAKURA Internet Inc.
[†2] 株式会社はてな Hatena Co., Ltd.
[a)] y-tsubouchi@sakura.ad.jp

とは呼べないほどの小型のラック群をあらゆる場所に分散して配置する分散型データセンターの構想が報告されている．分散型データセンターにおける各データセンターは単に分散するだけでなく，それぞれが協調し，全体としては一つとして見えるように動作することから，本稿では，分散型データセンターの命名を改め，超個体型データセンターとする．このようなコンピューティング基盤の変遷から，システムの分散化がさらに進み，システムの規模と複雑性が今後高まっていくと予想できる．

本報告では，ホスト上で動作し，ネットワークソケット宛ての要求を処理するプロセスをネットワークサービスと呼ぶ．ネットワークサービスが相互にネットワーク通信することにより，インターネットサービスが構成されるため，ホストの故障や性能劣化により，他のネットワークサービスに影響することがある．システムを変更するときに，その変更に問題があった場合に変更箇所に依存する全てのネットワークサービスへ影響が伝搬する可能性があるため，サービス同士のネットワーク依存関係を知らずに変更すると，システム管理者の予想よりも大きな範囲に障害が発生することがある．したがって，システム管理者はサービス間の依存関係を把握することにより，変更が影響する範囲を特定できることが重要となる．しかし，あるネットワークサービスが異なる別のサービスにネットワーク接続するには，通常，ネットワークサービスプログラムのソースコードや設定ファイルに接続先を静的に記述するため，あらゆる場所に接続のための設定が書かれていることになる．そのため，システム管理者は，見落としがないように未知のネットワークサービスの依存関係を網羅するために手間を要することになる．したがって，実際のネットワーク接続を追跡することにより，システム管理者の手によらず，自動で未知のネットワークサービスの依存関係を発見する必要がある．

未知のネットワークサービスの依存関係を発見するために，いくつかの先行手法 [7], [8], [9], [10] が提案されている．これらの手法は，ネットワークトラヒックを受動的に観測し，相関する活動を示すネットワークサービスを発見することから，アプリケーションに依存せずに，依存関係を発見できる．しかし，本来依存がないネットワークサービスに依存があると誤判定してしまう比率（偽陽性率）が高いことと，誰が誰に依存するのかという依存関係の方向を識別できないという課題がある [11]．

未知のネットワークサービスとの通信を発見するためには，OS の機能により網羅的に接続情報を取得できるトランスポート通信の接続を終端するホスト上で検出すればよい．また，偽陽性を削減するために，ネットワークサービス間のトラヒックの相関性を観測するのではなく，実際にホスト上で発生したサービス間の接続を直接検出すればよい．さらに，依存関係の方向を識別するために，トランスポート接続の両端のプロセスを，接続を要求する側と接続を待ち受ける側に分け，前者が後者に依存すると判定すればよい．そこで，本研究では，ネットワーク通信を終端する各ホスト上で確立されたトランスポート接続情報を網羅的に収集することにより，ネットワークサービス間の依存関係を自動で追跡可能な基盤を提案する．この追跡基盤では，各ホスト上で検出したトランスポート接続情報を，接続先および接続元のネットワークサービスのプロセス情報を含めて，中央のデータベースに定期的に送信することにより，システム管理者または各ホストが中央のデータベースに問い合わせ，特定のネットワークサービスの依存関係を取得する．

本研究で提案する基盤を実現することにより，まず，未知のネットワークサービスであっても，OS がサポートするトランスポート接続を利用する限りは，依存関係を追跡できる．次に，トランスポート通信のうち広く利用されている TCP および UDP の接続情報とサービスのプロセス情報を OS が提供するため，既存のサービスを変更せずに依存関係を追跡できる．さらに，ネットワークサービスのプロセス情報を含むために，ホスト単位の依存関係ではなく，より小さな粒度であるホスト上のプロセス単位の依存関係を追跡できる．加えて，中央のデータベースに現在から過去に渡って接続情報を保存するために，特定の時刻や一定間隔で短時間の間実行されるプロセスにより一時的に接続される依存関係を追跡できる．最後に，各ホストが自律的に自身の依存関係を収集しつつ，自身の依存関係を中央のデータベースから取得できる．

本論文を次のように構成する．2 章では，ネットワークサービスの依存関係の発見に関する先行研究を整理する．3 章では，2 章で整理した課題を解決するための提案手法を述べる．4 章では，本論文をまとめ，今後の展望を述べる．

2. ネットワークサービスの構成と依存

システムが大規模化かつ複雑化している状況において，システムを構成するネットワークサービス間の依存関係をシステム管理者が把握することが難しくなっているため，システムの変更時の影響範囲を特定できず，予想を超える規模の障害に発展することがある．本章では，ネットワークサービスの分散構成とネットワークサービスの依存関係を自動で発見するための技術を整理する．

2.1 ネットワークサービスの分散構成

インターネットサービスは，小さく再利用可能な複数のモジュールを合成して，実現されている．例として，図 1 にインターネットサービスのうちの 1 つである Web サービスの典型的な構成を示す．Web サービスは，Web サーバ，アプリケーションサーバ，データベースサーバの 3 階層のネットワークサービスを基本として構成され

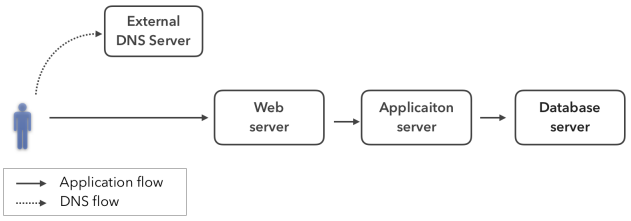

図 1 Web サービスの 3 階層構成

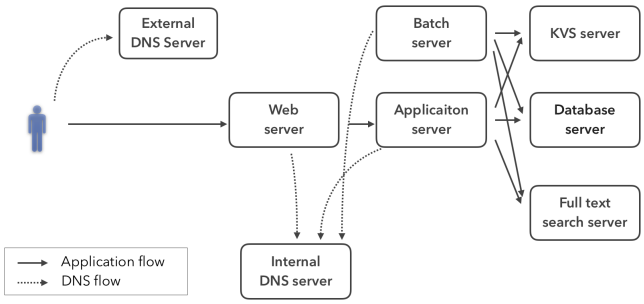

図 2 Web サービスの分散構成

る [12]．この基本構成に加えて，Web サービスの一部機能を効果的に実装するために，専用のミドルウェアを追加した構成を図 2 に示す．まず，アプリケーションサーバから，リレーショナルデータベース（RDB: Relational Database）サーバ，キーバリューストア（KVS: Key-Value Store）サーバ [13], [14], [15] や全文検索サーバ [16], [17] などの NoSQL[18] を利用する．次に，特定の時刻や時間間隔でアプリケーションを実行する場合，バッチサーバ上のタイマーがアプリケーションを実行する．Web サービスの利用者は，Web サービスにアクセスするために，外部の DNS サーバに問い合わせ，Web サーバの IP アドレスを取得する．内部のアプリケーションサーバやバッチサーバ上のアプリケーションからデータベースサーバや全文検索サーバに接続するときに，ドメイン名を利用する場合，内部 DNS サーバに問い合わせ，データベースサーバの IP アドレスを取得する．

1 台のサーバの処理能力には限界があることから，Web サービスのアクセスが増加すると，増加量に応じて負荷を複数のサーバに分散する必要がある．Web サーバやアプリケーションサーバは，ロードバランサを利用することにより，複数の同一の Web サーバやアプリケーションサーバへ負荷を分散する．RDB サーバや NoSQL サーバについては，同一のデータを複数のサーバに配置するレプリケーションを利用し，問い合わせを複数のサーバへ分散させる．また，特定のデータを特定のサーバに配置することにより，データベースの処理を複数のサーバへ分散させることもある．

Web サービス企業では，単一の企業で複数の Web サービスを提供することがあるため，あるサービスの機能を他のサービスから利用するために，サービス間をネットワーク接続することがある．ネットワーク接続の構成の方法は複数の手段があり，例えば，図 2 における Web サーバから異なる別のサービスの Web サーバへ接続することがある．また，単一の巨大なサービスを複数の小さなサービスに分割した上で，疎結合な状態にして各サービスの再利用を高めた構成をとるマイクロサービスアーキテクチャ [19] が注目されている．

2.2 ネットワークサービス間の依存関係

我々は，ネットワークサービス間の依存関係を，Zand らの研究 [11] と同様に，次のように定義する．サービス S_1 での遅延，性能低下および故障が，直接または間接的に，サービス S_2 の遅延，性能低下および故障を引き起こすのであれば，サービス S_2 はサービス S_1 に依存する．Chen らの研究 [8] は，ネットワークサービス依存関係を local-remote 依存と remote-remote 依存の 2 つに分類している．S_1 がクライアントにサービスを提供するために S_2 に接続する必要がある場合，サービス S_1 はサービス S_2 に local-remote 依存関係をもつ．サービス S_2 に接続するために，遠隔にあるクライアントがサービス S_1 に最初に接続する必要があるのであれば，サービス S_2 はサービス S_1 に remote-remote 依存関係をもつ．

例えば，クライアントである Web サーバから要求を受けたアプリケーションサーバは，データベースサーバに問い合わせ，結果を取得したのちに，Web サーバへ応答を返却するため，アプリケーションサーバはデータベースサーバに local-remote 依存関係をもつ．一方で，Web サーバに接続するクライアントは，まず DNS サーバに接続し，IP アドレスを取得したのちに，Web サーバに接続するため，Web サーバは DNS サーバに remote-remote 依存関係をもつ．

2.3 ネットワークサービスの依存関係の発見

ネットワークサービス間の依存関係の発見のための手法は，大きく分けて，外部からシステムの振る舞いを観測する観測志向のアプローチと，アプリケーション処理やネットワーク通信の間に計測点を設定することにより，要求処理の一連の流れの中で計測点を通過させる介入志向のアプローチに分類できる．

2.3.1 観測志向アプローチ

観測志向のアプローチとして，ネットワークトラヒックの振る舞いを観測する Sherlock[7], Orion[8], Macroscope[9], NSDMiner[10], [20] がある．Sherlock[7] は，異なるサービス間のパケットの時間相関を利用する．Orion は，トラヒックの遅延分布のスパイクを利用する [8]．NSDMiner は，ローカルのネットワークサービスが要求されていると仮定して，リモートのサービスが要求されている確率を計算することにより，local-remote 依存を発見する [10]．

いずれの手法においても，まず，トラヒックの相関性を

観測するものであり，依存関係を発見する決定的な手法ではないため，偽陰性と偽陽性がある．トラヒックに関して，アプリケーション層の情報を考慮しないことから，どのトラヒックが依存関係に起因するのか，それとも単に偶然の一致により，依存関係があるようにみえるかを区別することは困難である．また，相関を観測することから，依存関係の方向を識別することが難しい．さらに，トラヒックから全体的に依存を抽出することから，ほとんど利用されないネットワークサービスへの依存を見逃しうる．

Macroscope は，ホストまたはネットワークデバイスで収集するネットワークパケットとホスト上のトランスポート接続情報を結合する [9]．ホスト上のトランスポート接続情報とアプリケーションプロセスの情報を利用することにより，偽陽性を低減していることが特徴である．しかし，クライアント PC においてどのようなネットワークサービスを利用しているかを調査するための手法であり，サーバサイドの依存関係の発見については，今後の課題とされている．

2.3.2 介入志向アプローチ

介入志向アプローチとして，Pinpint[21]，Magpie[22]，X-Trace[23]，Dapper[24]，Rippler[11]，Clawson の研究 [25] およびサービスメッシュ [26] がある．Pinpoint，Magpie，X-Trace は，アプリケーションの処理中に，リクエストの識別子をネットワークプロトコルに埋め込み，後続の依存するシステムへ識別子を伝搬させる [21], [22], [23]．Dapper は，リクエストの識別子を埋め込むオーバーヘッドをサンプリングにより低減させている [24]．Dapper のモデルにしたがったオープンソースの実装として，Apache HTrace[27]，OpenZipkin[28]，Jaeger[29] などがあり，これらの手法をまとめて分散トレーシングと呼び，分散トレーシングの仕様を標準化する OpenTracing[30] が登場している．Rippler は，ネットワークトラヒックに遅延を注入することにより，依存するシステムへ遅延が伝搬する特性を利用する [11]．Clawson の研究は，ホスト上の OS 内のパケット処理部分に計測点を設定し，パケットの通過をログとして記録する [25]．サービスメッシュは，マイクロサービス [19] やネットワークサービスの単位でプロキシを配置し，プロキシを経由してその他のサービスと通信させることにより，アプリケーションを変更することなく，依存するサービスの依存を発見し，トラヒックを制御する [26]．サービスメッシュの実装として Istio[31] と Linkerd[32] があり，プロキシのみの実装として Envoy Proxy[33] がある．

いずれの手法も，計測点を明示的に設定しているため，偽陽性が発生しづらいという利点がある．しかし，アプリケーションの一連の処理に介入するため，本来のアプリケーション処理に加えてオーバーヘッドが発生する．また，X-Trace[23] と Dapper[24] は，識別子を埋め込むことから，アプリケーションの変更が必要となる．Rippler[11] は，各ネットワークサービスが特定の時間枠で遅延するため，あらかじめ学習中のサービスのセットを知っておく必要がある．Clawson の研究 [25] は，全てのパケットが設定した計測点を通過するようにすることで，未知のネットワークサービスの依存を発見できる．同様に，サービスメッシュ [26] は，サービス単位のプロキシを透過型プロキシとして利用することにより，未知のネットワークサービスの依存を発見できる [34]．しかし，Clawson の研究とサービスメッシュ以外の手法は，既知のトラヒックや要求処理に計測点を設定する必要があるため，未知のネットワークサービスの依存を発見できない．

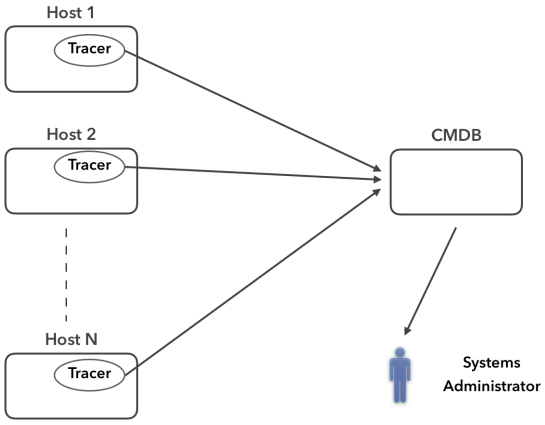

図 3 提案手法の基盤構成

3. 提案手法

既存手法の特徴を踏まえると，依存関係の検知精度を高めつつ，依存の方向を識別した上で，未知のネットワークサービスの依存関係を発見するためには，ネットワーク通信を終端する各ホスト上で確立された接続情報を網羅的に収集する基盤が必要となる．網羅的に収集するには，観測のためのオーバーヘッドを小さくすることが望ましいため，アプリケーション処理に介入しない手法が求められる．

3.1 提案手法の詳細

以下に提案手法の要件を示す．

(1) OS の機能により，ホスト上の全ての接続情報を取得できるトランスポート通信の接続ログをアプリケーション処理に介入することなく収集できる
(2) トランスポート接続の両端のプロセスを，接続を要求する側と接続を待ち受ける側に分けられる
(3) 接続ログを永続保存し，指定した時間範囲で検索できる

図 3 に，各要件を満たす基盤の構成を示す．まず，トレーサーを各ホスト上で動作させ，トレーサーが OS からトランスポート接続情報を定期的に収集する．次に，トレーサーは自身のホストの OS から待ち受け状態のポート一覧を取得し，当該ポートの接続の相手先サービスから

当該ポートで待ち受けるサービスに依存すると判定する．さらに，トレーサーは，接続情報に判定した依存の方向を含め，中央の CMDB（接続管理データベース）へ送信する．最後に，システム管理者または各ホストが，ホストやネットワークサービスを一意に識別する情報と時間範囲を CMDB に問い合わせ，対象に紐づく依存関係を取得する．

提案手法における接続情報は，送信元と送信先の IP アドレスとポート番号にネットワークサービスのプロセス ID とプロセス名を加えた，$(< src\_ip >, < src\_port >, < dst\_ip >, < dst\_port >, < pid >, < process\_name >)$ の 6 タプルで表現する．$< pid >$ と $< process\_name >$ については，接続情報を収集するホスト上のプロセス名を指す．

ネットワークサービス間の依存の方向を識別するために，ネットワークサービス間のトランスポート接続について，次のように，接続を要求する側と接続を待ち受ける側に分けた上で，依存の方向を決定する．ホスト X 上のネットワークサービス A がポート M で待ち受けているとして，ホスト Y 上のサービス B がホスト X 上のポート M に対して接続するとすると，サービス A がなければサービス B は接続処理に失敗するため，サービス B はサービス A に依存すると言える．

ホスト上で取得可能な全てのトランスポート接続情報を収集すると，接続情報の個数が大きくなり，CMDB 上のレコード数が大きくなるという問題がある．そこで，レコード数を削減するために，ネットワークサービス間の依存関係を識別するのみであれば，冗長となる接続情報を削減する．ネットワークサービスを構成する各プロセスやスレッドが OS から割り当てられるランダムな送信元ポートであるエフェメラルポート（Ephemeral Port，短命ポート）を利用して，別のネットワークサービスへ接続することがある．したがって，特定のポートで待ち受けるネットワークサービスに対して，特定のネットワークサービスのクライアントから複数のランダムな送信元ポートを利用して接続されることがあると言える．しかし，依存関係を識別するのみであれば，接続の度に異なるポート番号が割り当てられるエフェメラルポートを一意な接続情報とみなす必要がないため，トレーサーが特定のホストの特定のポートに対するエフェメラルポートによる接続情報を重複するものとみなし，それらを集約した上で，CMDB へ送信する．

3.2 提案手法の制約

提案手法は，OS のプロセス単位の依存関係を発見できる一方で，ネットワークサービスに対する要求単位の依存関係を追跡できない．要求単位の依存関係を追跡するためには，X-Trace[23] や Dapper[24] のように，アプリケーション処理に介入し，要求を識別するための識別子を埋め込む必要がある．ローカルホスト上のプロキシなどの接続を委譲するネットワークサービスにより，実際の接続元のプロセスの情報を隠蔽する可能性がある．

提案手法の実装は，ホスト上の確立済みの接続のスナップショットを定期的に取得することから，取得する時間間隔以内に終了する接続を発見できない可能性があるため，偽陰性がある．

4. まとめと今後の展望

本研究では，超個体型データセンターに向けて，ホスト上のトランスポート接続の情報を収集することにより，未知のネットワークサービスを含むサービス間の依存関係を自動で発見する手法を提案した．

今後の展望として，まず，提案手法の実装を進め，実験をした上で偽陽性と偽陰性について，評価する予定である．次に，コンテナ型仮想化環境において動作してネットワークサービスの依存関係を追跡できるようにしていくつもりである．さらに，依存関係をリアルタイムに収集できる提案手法の特徴を活かし，システム管理者が意図していない依存の検知によるマルウェア検出や，本来存在するはずの依存を発見できない場合に異常とみなすような異常検知への応用を進める．最後に，エッジコンピューティングや小・中規模データセンターにおいて，エッジやデータセンター間の依存関係のようなネットワークサービスよりも大きな粒度における依存を発見する手法を考えていく必要がある．

参考文献

[1] J Lin, W Yu, N Zhang, X Yang, H Zhang, and W Zhao: A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications, *IEEE Internet of Things Journal*, Vol. 4, No. 5, pp. 1125–1142 2017.

[2] M Aazam and E N Huh: Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT, *IEEE International Conference on Advanced Information Networking and Applications*, pp. 687–694 2015.

[3] B Kashif, K Osman, E Aiman, and S U Khan: Potentials, Trends, and Prospects in Edge Technologies: Fog, Cloudlet, Mobile Edge, and Micro Data Centers, *Computer Networks*, Vol. 130, pp. 94–120 2018.

[4] 山口弘純, 安本慶一：エッジコンピューティング環境における知的分散データ処理の実現，電子情報通信学会論文誌 B， Vol. 101, No. 5, pp. 298–309 2018.

[5] M Satyanarayanan: The Emergence of Edge Computing, *Computer*, Vol. 50, No. 1, pp. 30–39 2017.

[6] 松本亮介, 坪内佑樹, 宮下剛輔：分散型データセンター OS を目指したリアクティブ性を持つコンテナ実行基盤技術，情報処理学会研究報告インターネットと運用技術（IOT），Vol. 2019-IOT-44, No. 27, pp. 1–8 2019.

[7] P Bahl, R Chandra, A Greenberg, S Kandula, M Srikanth, D A Maltz, and M Zhang: Towards Highly Reliable Enterprise Network Services via Inference of Multi-Level Dependencies, *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 4, pp. 13–24 2007.

[8] X Chen, M Zhang, Z M Mao, and P Bahl: Automat-

ing Network Application Dependency Discovery: Experiences, Limitations, and New Solutions, *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 117–130 2008.

[9] P Lucian, B-G Chun, I Stoica, J Chandrashekar, and N Taft: Macroscope: End-Point Approach to Networked Application Dependency Discovery, *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 229–240 2009.

[10] A Natarajan, P Ning, Y Liu, S Jajodia, and S E Hutchinson: NSDMiner: Automated Discovery of Network Service Dependencies, *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2507–2515 2012.

[11] A Zand, G Vigna, R Kemmerer, and C Kruegel: Rippler: Delay Injection for Service Dependency Detection, *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2157–2165 2014.

[12] X Liu, J Heo, and L Sha: Modeling 3-tiered Web Applications, *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 307–310 2005.

[13] G DeCandia, D Hastorun, M Jampani, G Kakulapati, A Lakshman, A Pilchin, S Sivasubramanian, P Vosshall, and W Vogels: Dynamo: Amazon' s Highly Available Key-value Store, *Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, pp. 205–220 2007.

[14] F Chang, J Dean, S Ghemawat, W C Hsieh, D A Wallach, M Burrows, T Chandra, A Fikes, and R E Gruber: Bigtable: A Distributed Storage System for Structured Data, Vol. 26, No. 2, pp. 4:1–4:26 2008.

[15] A Lakshman and P Malik: Cassandra: A Decentralized Structured Storage System, *ACM SIGOPS Operating Systems Review*, Vol. 44, No. 2, pp. 35–40 2010.

[16] C Gormley and Z Tong: *Elasticsearch: The Definitive Guide: A Distributed Real-time Search and Analytics Engine*, O'Reilly Media, Inc. 2015.

[17] T Grainger and T Potter: *Solr in Action*, Manning Publications Co. 2014.

[18] J Han, E Haihong, G Le, and J Du: Survey on NoSQL database, *2011 Sixth International Conference on Pervasive Computing and Applications (ICPCA)*, pp. 363–366 2011.

[19] M Fowler and J Lewis: Microservices, http://martinfowler.com/articles/microservices.html.

[20] B Peddycord III and P Ning, and S Jajodia: On the Accurate Identification of Network Service Dependencies in Distributed Systems, *Large Installation System Administration Conference (LISA)*, pp. 181–194 2012.

[21] M Y Chen, E Kiciman, E Fratkin, A Fox, and E Brewer: Pinpoint: Problem Determination in Large, Dynamic Internet Services, *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 595–604 2002.

[22] P Barham, R Isaacs, R Mortier, and D Narayanan: Magpie: Online Modelling and Performance-aware Systems, *17th Workshop on Hot Topics in Operating Systems (HotOS)*, pp. 85–90 2003.

[23] R Fonseca, G Porter, R H Katz, S Shenker, and I Stoica: X-Trace: A Pervasive Network Tracing Framework, *USENIX Conference on Networked Systems Design & Implementation (NSDI)*, pp. 20–20 2007.

[24] B H Sigelman, L A Barroso, M Burrows, P Stephenson, M Plakal, D Beaver, S Jaspan, and C Shanbhag: Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Technical report, Google 2010.

[25] J K Clawson: Service Dependency Analysis via TCP/UDP Port Tracing, Master's thesis, Brigham Young University-Provo 2015.

[26] W Li, Y Lemieux, J Gao, Z Zhao, and Y Han: Service Mesh: Challenges, State of the Art, and Future Research Opportunities, *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 122–1225 2019.

[27] Apache HTrace, http://incubator.apache.org/projects/htrace.html.

[28] OpenZipkin, https://zipkin.io/.

[29] Jaeger, https://www.jaegertracing.io/.

[30] The OpenTracing project, https://opentracing.io/.

[31] Istio, https://istio.io/.

[32] Linkerd, https://linkerd.io/.

[33] Envoy Proxy, https://www.envoyproxy.io/.

[34] Proxy redirection - istio/istio Wiki, https://github.com/istio/istio/wiki/Proxy-redirection.