Vol.2021-OS-153 No.9
2021/7/20

高水準なマイクロサービス層における複数ドメインを連携させたインタークラウドHPC環境実現の検討

杉木 章義[1,a)]

概要：本研究では，Service Mesh を活用したマイクロサービス層におけるインタークラウド HPC 実現の可能性を検討する．従来のインタークラウドでは，各大学や研究機関，パブリッククラウド事業者が提供する仮想マシンや物理マシン，ストレージなどの主に計算機層を連携してサービスを実現することを想定していた．これに対して本研究では，近年，Web 分野で実用化が進んでいる Service Mesh を活用し，各学術機関が提供する Kubernetes や仮想マシンの上で稼働するマイクロサービスを連携させることで，HPC サービスを実現することを検討する．本研究では，L7 プロキシの Envoy，クロスドメイン認証連携のための SPIFFE/SPIRE，SINET5 の L2VPN サービスを活用し，その可能性や今後の課題を明らかにする．

Towards an Inter-cloud HPC Environment by Linking Multiple Domains at Higher-level Microservice Layer

1. はじめに

近年，HPC (High Performance Computing) 分野では，HPCI（革新的ハイパフォーマンス・コンピューティング・インフラ） [1] や JHPCN（学際大規模情報基盤共同利用・共同研究拠点） [2] など，複数の大学や学術機関を連携させた大型計算機システム運用が行われている．また，2021 年 3 月からデータ活用により一層フォーカスしたデータ活用社会創成プラットフォーム mdx [3] の運用も開始されている．

このような状況の中で，人手による制度や運用面の支援に限らず，計算機のシステムソフトウェア面からの支援が必要である．HPCI のストレージ面に関しては，Gfarm をベースとした HPCI 共用ストレージが 3 拠点の連携で長期にわたり運用されている．認証面に関しても，HPCI GSI 認証や，国立情報研究所が提供する学術認証フェデレーション GakuNin による Shibboleth 認証が提供されている．一方の計算機面に関しては，スーパーコンピュータの時間トークンとしてや，仮想や物理マシンとして提供されるものが多く，ソフトウェア環境の整備や複数拠点間の連携方法はこれまで大きく利用者に委ねられていた．

計算機環境の整備では，クラウドの普及を背景にインタークラウドが過去に検討された．しかしながら，インタークラウドでは仮想マシンや仮想ネットワークの連携による一体的な計算環境の実現が中心的話題であり，より上位のアプリケーション層の支援や，各拠点が提供する複数のアプリケーション相互の連携までは，あまり強く踏み込んではいなかった．

近年，Web の分野では産業界が先行し，Docker や Kubernetes [4] などによる Web サービスのコンテナ化が進んでいる．また，一つの巨大なサービスを小規模なコンテナに分割し，それらの疎結合でサービス全体を実現するマイクロサービス [5] 化が進行している．さらには，マイクロサービスの実現を容易にするために，Istio [6]，Consul，Linkerd などの Service Mesh が登場している．Service Mesh はコンテナで実現されたサービスのディスカバリや相互接続，mTLS (mutual-TLS, 2-way TLS) などによる通信の暗号化，負荷分散やトラフィック制御，障害対策などをサービス本体から切り離し，サービス本体に付随するサイドカーコンテナとして透過的に実現することで，マイクロサービスの実現を用意にしている．また，Service Mesh では，Control Plane と Data Plane を分離して，中央集権的にトラフィックを制御可能としている点も大きな特徴である．

[1] 北海道大学情報基盤センター
Information Initiative Center, Hokkaido University, Sapporo 060–0811, Japan
[a)] sugiki@iic.hokudai.ac.jp

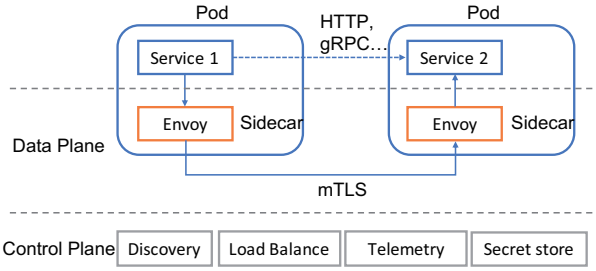

図 1　マイクロサービスと Service Mesh
Fig. 1　Microservices and Service Mesh

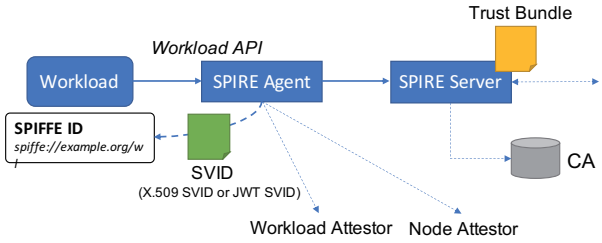

図 2　SPIFFE/SPIRE のアーキテクチャ
Fig. 2　SPIFFE/SPIRE Architecture

Envoy [7] は L7/L4 層を対象としたプロキシであり，Istio などの Data Plane の実装に広く用いられている．

本研究では，Service Mesh を活用したマイクロサービス層におけるインタークラウド HPC 連携の可能性を検討する．各学術機関で開発された HPC アプリケーションをマイクロサービスとして実現し，Service Mesh により複数の学術機関が提供するマイクロサービスを結合し，より大きな HPC サービスを実現することを目標とする．組織（ドメイン）の異なるマイクロサービス間の連携には大きな課題があり，本研究ではその可能性と課題を明らかにする．

2. 関連技術

2.1 マイクロサービスと Service Mesh

マイクロサービスはソフトウエア開発技法の一つであり，アプリケーションを非常に小さい多数のサービスに分割し，それらの疎結合で巨大なサービスを実現する方式である，サービスを構成する各コンポーネントの独立性を高め，それぞれの開発や更新を容易にすることで，全体の品質の継続的な改善を目指すアプローチである．マイクロサービスは，Netflix や Amazon など国内外の多数の企業で導入され，実運用が行われている．

マイクロサービスの実現には，いくつかの共通の技術基盤が存在する．まず，マイクロサービス間の相互通信には，**HTTP** や **gRPC** など標準的なプロトコルが用いられる．標準的なプロトコルを採用することで，各マイクロサービスを C/C++や Java，JavaScript，Python，Go など独立した言語で開発して，相互に連携させることが可能となる．また，マイクロサービス間通信の安全性をさらに高めるために，**mTLS** による通信暗号化が近年，広く用いられる．従来の TLS がクライアント側からサーバ側の証明や検証のみであったのに対して，mTLS はクライアント・サーバ双方の検証を行い，なりすまし攻撃などを抑止する．さらに，暗号鍵の定期的なローテーションも多くのシステムで導入されている．

Service Mesh は，マイクロサービスの実現に必要な多くの機能を提供するソフトウェア基盤である（**図 1**）．Service Mesh の実装として，Istio，Consul，Linkered などが知られており，パブリッククラウドでも，Amazon Web Services (AWS) の AppMesh，Google Cloud Platform (GCP) の Anthos Service Mesh などがマネージドサービスとして提供されている．

Service Mesh では，マイクロサービスのディスカバリ，相互の暗号化通信，負荷分散，耐障害性，計測や監視など，マイクロサービス開発者が実装上の負担となる機能を代わりに提供する．Service Mesh では，転送先を決定し，転送制御を行う **Control Plane** と実際の転送処理を行う **Data Plane** を分離する．Data Plane の実装では，Istio や Consul，AppMesh などの多くの Service Mesh で Envoy が非常によく用いられている．**Envoy** は C++で実装された高性能かつ高機能なレイヤ 7 のプロキシであり，Kubernetes では Pod の**サイドカーコンテナ**としてマイクロサービス本体のコンテナに付随して挿入され，透過的に通信の中継や制御を行う．

さらに，多くの Service Mesh では Kubernetes に限らず，仮想マシンの上で直接構築されたマイクロサービスにも対応する．Kubernetes とそれらの相互の透過的な通信も可能とするものが多く，高い柔軟性がある．

2.2 SPIFFE/SPIRE

標準規格としての **SPIFFE** [8]，およびリファレンス実装としての **SPIRE** は，Service Mesh に対して，サービスの識別・認証基盤を提供するフレームワークである（**図 2**）．SPIFFE 規格は SPIRE に限らず，Istio や Consul などの Service Mesh でも対応している．

SPIFFE の規格では，マクロサービスを**ワークロード**，それらのサービスが稼働する計算機を**ノード**と読み替える．ワークロードの特定や識別には，URL 形式で表現された **SPIFFE ID** を用いる．例として，SPIFFE ID は次のように表記される．

```
spiffe://hokudai.spire/path/to/workload
```

SPIFFE ID は上記のように大きく **Trust Domain** 部とワークロードを特定するパス部の二つで構成される．なお，SPIFFE ID の Trust Domain は DNS のドメイン名と必ずしも一致させる必要はなく，相互に連携する機関の約束で自由に設定することができる．

SVID (SPIFFE Verifiable Identify Document) は，

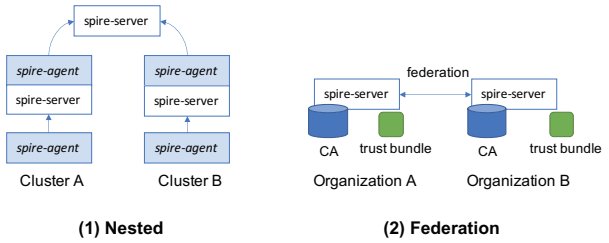

図 3　フェデレーションの種類
Fig. 3　Types of Federation

SPIFFE ID の真偽性を検証するために，情報の相互交換で用いられるドキュメントである．現在の SVID では，X.509 証明書を基礎とした **X.509 SVID** と，Web の API 連携で用いられる JWT トークン形式をもとにした **JWT SVID** の二つが規格化されている．各ワークロードは，**Workload API** を通じて，SVID の取得や検証などを行う．

Trust Domain となる組織では，それぞれ独立に **CA**（Certification Authority，認証局）を運用する．Trust Domain 間で，**Trust Bundle** を通じて相互に認証を行い，フェデレーションを実現する．連携の方法としては，主に**図 3** に示す二つの形態がある．(1) **ネスト構成**では，二つのクラスタ間で共通のルート CA を設置し，信頼のチェーンを確立し，相互認証可能とする．(2) **フェデレーション構成**では，異なる運用組織間で Trust Bundle を交換し，独立性を保ちながら，相互に識別や認証を行う．

SPIFFE の公式リファレンス実装として，**SPIRE** が提供されている．SPIRE は，SPIRE サーバと，SPIRE エージェントの二つで構成されている．SPIRE では，ワークロードを稼働させるノードを検証する **Node Attestor** と，ワークロードそのものを検証する **Workload Attestor** によるプラットフォーム・サービス相互の検証により，安全性が担保されている．

本研究では，SPIRE と Envoy を使用し，Service Mesh を構築する．また，認証後の通信制御に相当する認可に関しては，公式ドキュメント [9], [10] を参考に **OPA** (Open Policy Agent) [11] を使用する．OPA ではポリシーを Rego 言語で記述する．

当初，広く普及している Istio の使用を検討したが，公式の Istio の実装では，同じ Trust Domain に所属する異なるクラスタ間の連携のみ考慮されており，異組織間のフェデレーション構成は現時点でサポートされていない．なお，Istio と SPIRE を組み合わせた実現に関しては，IBM で研究が進められている [12], [13]．本機能の検証に関しては，将来的な課題とする．

3. 提案

本研究では，Service Mesh の機能を活用し，マイクロサービス層におけるインタークラウド HPC の実現可能性を検討する．本プロジェクトでは，Web 分野において現時点で実現されているクラウド技術をもとに，その一部の機能を HPC 分野に移植や転用することで，HPC 分野で伝統的に使用されているジョブスケジューラを中核としたソフトウェア環境の改善と融合を目指す．

3.1 実現上の課題

本研究では以下の問題解決を長期的な目標とし，解決のための方針のいくつかを本論文で示す．

- **マイクロサービス化する HPC アプリケーション：** 最初の課題として，各拠点で研究開発が進められている HPC 分野のアプリケーションを各機能ごとにどのようにマイクロサービスとして分割し，提供していくかという課題がある．多くの伝統的な HPC アプリケーションは，バッチスケジューラに投入するバッチジョブの形態で実現されており，継続的に稼働するサービスとして実現されていない．また，同様の理由で CI/CD (Contanious Integration/Contanious Delivery) の HPC 分野への導入にもギャップがある．一方で，気象分野や地震などへの防災分野など，ミッションクリティカルな分野で HPC アプリケーションのサービス化が実現されている．また，学術研究のより一層の発展のために，単なる計算機の提供から，より上位となる研究支援サービス提供への移行が必要である．また，Web 分野に比較的近い大規模データ（ビッグデータ）の収集や解析，機械学習サービス，数値計算のプレ処理・ポスト処理などはマイクロサービス化が容易で，早期に実現すべきだと考えられる．
- **暗号化によるオーバヘッド：** Service Mesh では，一般的に mTLS による暗号化通信が広く用いられている．mTLS は通信内容の暗号化や接続端点の検証により，セキュリティを高める一方で，通信上のオーバヘッドが課題となる．特に大容量データの転送を行う HPC 分野のアプリケーションでは大きな課題となる．また，多くの Service Mesh では，Envoy などのサイドカーコンテナによる通信の横取りで，透過的に多くの機能を実現しており，そのソフトウェア処理のオーバヘッドも課題となる．
- **クロスドメイン認証連携：** 最後の課題として，いかに複数の学術機関の間でクロスドメイン認証を行い，フェデレーションを実現していくかという課題がある．組織間の手続き上の問題もあるが，技術的な課題の解決が必要である．

3.2 マイクロサービス導入の利点

一方で，HPC 分野へのマイクロサービス技術の導入は下記のようなメリットがある．

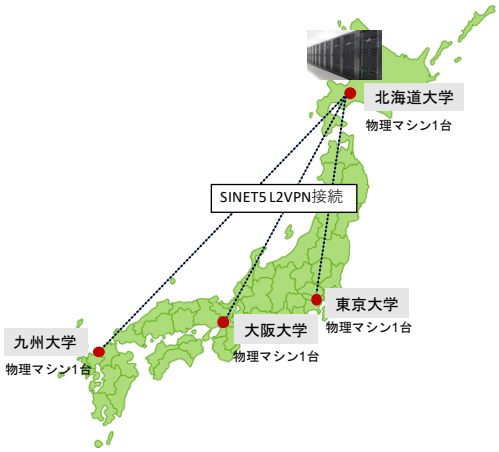

図 4　実験環境
Fig. 4　Experimental Setup

表 1　各拠点の計算機仕様
Table 1　Node Specification

部品	仕様
CPU	Intel Xeon Gold 6138 (20 コア/2.0GHz) × 2
メモリ	256GB
ストレージ	240GB SSD × 2
	2TB HDD × 2（遠隔 3 拠点のみ）
ネットワーク	25GbE × 2

- **サービスの標準化：** Service Mesh を HPC アプリケーションの提供基盤とすることで，各拠点の提供サービスが標準化され，その自由な組み合わせによる，より大きなサービスの構築が容易となる．
- **セキュリティ，安定性，耐障害性の向上：** 負荷分散やセキュリティ，耐障害性のための機能に関して，Service Mesh の機能をそのまま用いることで，これらの機能を透過的かつ容易に実装することが可能となる．
- **相互運用性の向上：** 組織間の連携には，依然として課題がある．SPIFFE/SPIRE では，フェデレーション認証のための仕組みが導入されており，課題の一部の解決が期待される．

4. 予備実験

4.1 実験環境

予備実験では，北海道大学情報基盤センターのインタークラウドパッケージ [14] を使用した（図 4）．北から順に，北海道大学，東京大学，大阪大学，九州大学の 4 拠点，各 1 台の合計 4 台の計算機を使用した．各拠点の計算機仕様を表 1 に示す．北海道大学とそれ以外の 3 拠点は SINET の L2VPN サービス [15] で接続されている，3 拠点の接続帯域は直上のスイッチでそれぞれ 40Gbps，20Gbps，10Gbps であるが，各拠点で経由するキャンパスネットワークの影響を受ける．

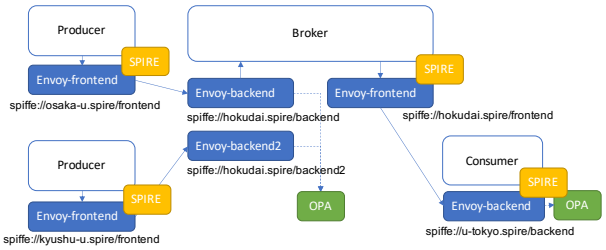

図 5　プロトタイプ環境の構成
Fig. 5　Organization of Prototype Testbed

4.2 ソフトウェア構成

インタークラウドパッケージは実際には単一 Trust Domain で構成された環境だが，仮想的に各拠点が独立したクロスドメイン環境を想定してソフトウェア環境を構築する．具体的には，それぞれ異なる独立した自己証明 CA 認証局を運用し，相互に Trust Bundle を交換し，JWT トークンを使用したフェデレーションにより，相互認証を実現する．

各拠点の物理マシンでは，kubeadm を使用し，Kubernetes を導入した．Kubernetes はシングルノード構成とし，マスタとスレーブを区別しない．本研究では，マイクロサービス，Envoy および SPIRE を全て Kubernetes 上で稼働させる．SPIRE 自身は Kubernetes に限らず，仮想・物理マシン上で直接，SPIRE を稼働させることも可能である．OpenStack 用の IID (Instance ID) Attestor なども第三者の貢献により提供されているが，検証は本研究の将来的な課題とする．Kubernetes の各ノード検証のための Node Attestor としては，`k8s_psat` を使用した．

各ソフトウェアのバージョンは，Kubernetes 1.21.1，SPIRE 0.12.3，Envoy 1.18.2，OPA 0.24.0 である．

4.3 SPIFFE/SPIRE 構成

検証環境の想定アプリケーションとして，SINET-Stream [16]，またはその実装で用いられている Kafka [17] を想定する．大阪大学と九州大学で収集した大規模データを北海道大学で運用する Kafka のブローカに集約し，その内容を東京大学に設置されたデータ活用社会創成プラットフォーム mdx に転送することを想定して，SPIRE を構成した．

本研究の SPIRE の構成を図 5 に示す．mTLS はクライアント・サーバの双方を検証することから，mTLS の接続ごとに，独立して暗号鍵や証明書の管理が必要である．現在の Envoy の実装では，複数の SDS (Secret Data Service) を参照できないことから，接続端点ごとに Envoy を設置する．各拠点の役割ごとに，送信専用の Envoy（frontend），受信専用の Envoy（backend）を構成する．Envoy は同一 Pod 内のサイドカーコンテナとして挿入されており，北海道大学の拠点では，ブローカと 3 つの Envoy を同一 Pod

表 2　iperf3 のスループット測定結果
Table 2　Throughput measurement with iperf3

拠点	ホスト上（直接）	Envoy（mTLS）
東京大学	1.63 Gbps	1.60 Gbps
大阪大学	1.16 Gbps	1.13 Gbps
九州大学	817 Mbps	812 Mbps

表 3　Kubernetes 設定ファイルの記述行数
Table 3　Lines of Kubernetes configuration files

拠点	YAML 記述量
北海道大学	484 行
東京大学	230 行
大阪大学	142 行
九州大学	142 行

内に閉じ込める．受信側の Envoy では，OPA を参照し，接続の認可を行う．今回の OPA のポリシー記述では接続元を区別せず，同一のポリシーを記述している．北海道大学では，接続元が大阪大学または九州大学の拠点の場合のみ，接続を許可する．東京大学に設置した計算機では，北海道大学から接続された場合のみ許可する．

SPIFFE ID は図 5 のように個別の Envoy ごとに発行し，Envoy の接続の際に異なる拠点から検証できるようにした．SPIFFE ID の Trust Domain 部は，実際の DNS のドメイン名と異なるものを採用した．一般的に Kubernetes 上のワークロードでは，`spiffe://{Trust Domain}/ns/{` 名前空間名 `}/sa/{` サービスアカウント名 `}` の形式を採用するが，今回は簡潔性からその慣例に従わなかった．また，北海道大学のブローカについては，3 つの SPIFFE ID を一つにまとめることも可能である．しかしながら，今回は接続の厳密性を優先して，個別に発行することとした．

4.4　iperf3 のスループット測定結果

予備的な性能評価として，Envoy を挿入した場合のオーバヘッドによる影響を測定する．iperf3 によるスループット測定結果を**表 2** に示す．iperf3 は標準オプションを使用し，TCP の単一ストリームで，60 秒間測定している．表 2 では，(1) 物理マシン上で直接 iperf3 を実行した場合と，(2) コンテナ内で Envoy を経由して測定した場合の二つの結果を比較している．

表 2 から，Envoy 間で確立する mTLS を経由した場合に，僅かにスループットが低下していることが分かる．しかしながら，性能低下は僅かであることから，多くのアプリケーションの実用的な場面で大きな問題とならないことが予想される．ホスト上で直接実行した場合にスループットが 1Gbps 前後と伸び悩んでおり，その理由については現在調査を進めている．また，拠点間の距離に応じて，スループットも低下している．

上記の実験では，Envoy の TCP Proxy フィルタを使用し，それほど複雑なメタデータを含まないことから OPA を有効にせず測定した．また，同一の SPIRE 構成で HTTP フィルタを使用し，OPA を有効にした状態で，接続性の機能検証も完了している．さらには，Envoy は L7 プロキシであることから，Kafa にも直接対応しており，上記のフィルタを Kafka Broker フィルタに置き換えれば，SINETStream や Kafka などのアプリケーション層での制御も可能となる．

4.5　設定の記述量

表 3 に Kubernetes の YAML 設定ファイルの記述量を示す．今回，OPA ポリシーの記述，および Envoy 設定の記述も ConfigMap で実現しており，上記の行数は全ての設定内容を含んでいる．

表 3 から，構成が複雑になるにつれて，また送信側（Frontend）よりも受信側（Backend）で，記述量が増加していることが分かる．特に，北海道大学では，三つの Envoy の記述と OPA ポリシーの記述を含むことから，記述が大きくなっている．

今回の検証環境では，通信の方向を二つの拠点から北海道大学へ，北海道大学から東京大学の拠点へとうまく制限し，記述量を削減しているが，さらに拠点数が増加した場合や，通信の方向を全ての拠点間で自由とした場合に，記述量が飛躍的に大きくなる．

5.　議論

予備実験の結果から，認証のスケーラビリティに課題が生じている．それぞれの学術機関が独立に CA 認証局を運用し，相互認証によるフェデレーションを確立すれば，拠点の組み合わせ数となり，現実的なスケーラビリティに上限がある．今回の検証環境ぐらいの規模が恐らく限界であると考えられる．一方でもう一つの連携方法であるネスト構成を採用し，学術認証フェデレーション基盤 GakuNin などのように，例えば，国立情報学研究所にルート CA を設置すれば，スケーラビリティの問題は避けられるが，独立性や安全性が低下する．

また，SPIRE で用いる mTLS は通信端点の相互で証明書の検証を行う必要があることから，mTLS ごとに独立した証明書ストアが必要である．現状の Envoy では，一つの証明書ストアしか参照できないことから，通信端点ごとに Envoy コンテナを用意する必要がある，この問題はコミュニティでも指摘されており [18]，パッチが提案され，既に実装が行われている [19]．しかしながら，本機能は秘密鍵や公開鍵を直接参照する場合に限られており，SPIRE を SDS として参照する場合には，まだ対応できていない．

また安全性に関しては，現在，`k8s_psat` Attestor を使用しており，各組織で運用されている Kubernetes が正しく運用されている必要がある．しかしながら，Kubernetes を直接サービスとして提供している学術機関は日本国内で

はまだ少なく，今回の検証環境のように利用者が構築した Kubernetes 環境を使用する場合も多い．他の Attestor とも組み合わせ，そのような環境で安全性を向上させる方法についても検討していく．

6. まとめ

本研究では，Service Mesh を活用したマイクロサービス層におけるインタークラウド HPC 環境の実現可能性を検討した．サービスの識別や認証に SPIRE，プロキシに Envoy を使用し，日本国内の 4 拠点に配備された Kubernetes を使用し，検証環境を構築した．現在，接続性に関する検証を終えており，今後，性能面や安全性，実際の HPC アプリケーションでの評価を進めていく予定である．

現状，学術機関同士のフェデレーションで実現した場合に認証のスケーラビリティに課題があることが分かっており，改善を進める予定である．また，性能面での改善も進める予定である．

謝辞 本研究は JSPS 科研費 20K11837「高性能計算技術とマイクロサービス化技術の融合に関する研究」の助成を受けたものです．

参考文献

[1] 一般財団法人高度情報科学技術研究機構：革新的ハイパフォーマンス・コンピューティング・インフラ（HPCI）. https://www.hpci-office.jp/.
[2] 8 大学構成拠点：学際大規模情報基盤共同利用・共同研究拠点（JHPCN）. https://jhpcn-kyoten.itc.u-tokyo.ac.jp/.
[3] 東京大学：データ活用社会創成プラットフォーム mdx を導入–9 大学 2 研究機関が共同運営しデータ活用の産学官連携・社会実装・研究を推進–. https://www.u-tokyo.ac.jp/focus/ja/press/z0310_00027.html.
[4] Linux Foundation: Kubernetes – Production-Grade Container Orchestration. https://kubernetes.io/.
[5] Lewis, J. and Fowler, M.: Microservices (2014). https://martinfowler.com/articles/microservices.html.
[6] Istio Authors: Istio: Simplify observability, traffic management, security, and policy with the leading service mesh. https://istio.io/latest/.
[7] Lyft, Inc.: Envoy Proxy. https://www.envoyproxy.io/.
[8] SPIFFE Authors: SPIFFE – A universal identity control plane for distributed systems. https://spiffe.io/.
[9] SPIFFE Authors: Deploying a Federated SPIRE Architecture. https://spiffe.io/docs/latest/architecture/federation/readme/.
[10] SPIFFE Authors: OPA Authorization with Envoy and X.509-SVIDs. https://spiffe.io/docs/latest/microservices/envoy-opa/readme/.
[11] CNCF: Open Policy Agent – Policy-based control for cloud native environments. https://www.openpolicyagent.org/.
[12] Chen, D., Shlomo, R. and Solomon, T.: Attesting Istio workload Identities with SPIFFE and SPIRE. https://developer.ibm.com/components/istio/articles/istio-identity-spiffe-spire/.
[13] IBM: GitHub – Istio Identities with SPIFFE/SPIRE. https://github.com/IBM/istio-spire.
[14] 北海道大学情報基盤センター：学際大規模計算機システム：インタークラウドパッケージ. https://www.hucc.hokudai.ac.jp/intercloud/wide-area/.
[15] 国立情報学研究所：学術情報ネットワーク SINET5 – L2VPN/VPLS. https://www.sinet.ad.jp/connect_service/service/l2vpn.
[16] 竹房あつ子，孫　静涛，藤原一毅，吉田　浩，合田憲人：IoT ストリームデータ処理のためのソフトウェアライブラリ SINETStream の開発，情報処理学会研究報告 2020-IOT-48 (19)，pp. 1–8 (2020).
[17] Kreps, J., Narkhede, N. and Rao, J.: Kafka: a Distributed Messaging System for Log Processing, *NetDB workshop '11* (2011).
[18] Envoy Community: Enhance envoy listener TLS to support multiple trust domain. https://github.com/envoyproxy/envoy/issues/9284.
[19] Yoneda, T.: Isolated multiple trust domain mTLS in Envoy and Istio (2021). https://speakerdeck.com/mathetake/isolated-multiple-trust-domain-mtls-in-envoy-and-istio.