CPSサービスの迅速な立ち上げに貢献する東芝IoT基盤サービスHABANEROTSにおけるサービスメッシュの活用

HABANEROTS Toshiba Industrial IoT Platform Service Enabling Rapid Start of CPS Services Utilizing Service Mesh Technology

内田　正之 UCHIDA Masayuki　　樋口　靖和 HIGUCHI Yasukazu

CPS（サイバーフィジカルシステム）サービスの開発や事業展開を推進していくには，サービスの迅速な立ち上げや継続的な改善が重要で，それに適した開発体制やプロセスなどの整備を行っていくことが求められている。この取り組みを支える技術の一つにマイクロサービスアーキテクチャー（MSA：Microservices Architecture）がある。MSAは，迅速な開発サイクルが求められるシステムに有効なシステムアーキテクチャーであるが，システムを構成するマイクロサービス（MS：Microservices）数の増加に伴い，開発コストや運用コストの増大を招くという問題がある。

東芝は，CPSサービス開発の効率化を目的とした東芝IoT（Internet of Things）基盤サービスHABANEROTS（旧名称はHabanero）を開発している。HABANEROTSでは，サービスメッシュ技術を活用してMSAの問題解決を図っている。また，サービスメッシュ技術の活用により，性能テストでのボトルネック調査の時間短縮などによる開発の効率化や，Web API（Application Programming Interface）のエラー発生頻度などの集計結果に基づく継続的なサービス改善につなげられる。

To promote the development of cyber-physical systems (CPS) services and expansion of CPS businesses, establishment of the appropriate development platform and processes is becoming essential in order to rapidly start services and continuously improve quality. As part of these activities, the microservices architecture (MSA) is playing an important role in the rapid development cycles of CPS services. However, higher development and operation costs are serious issues accompanying the increase in the number of microservices.

Toshiba Corporation is developing HABANEROTS, previously known as Habanero, an industrial Internet of Things (IoT) platform service utilizing a service mesh technology to overcome issues related to MSA. HABANEROTS makes it possible to enhance the efficiency of development by shortening the time required for investigating bottlenecks in performance tests, as well as to realize continuous improvements in service quality based on the aggregated results including the frequency of error occurrence in Web application programming interfaces (APIs).

1. まえがき

CPSのサービス開発や事業展開を推進していくためには，CPSサービスの迅速な立ち上げと継続的な改善が重要となる。迅速な開発を行うためには，再利用性の高いコンポーネントや，アジャイル開発のような開発スタイルの実行が可能な開発体制・プロセスなどの整備を行っていくことが求められる。

こうした取り組みを支える技術の一つとしてMSA（この特集のp.7–10及びp.27–30参照）がある。MSAは，アプリケーションをビジネス機能ごとの小さな単位に分割することで，リリース管理の容易さや，回復性，スケーラビリティーなどを高められる。一方で，システムを構成するMSの数が増加すると，MS間の通信が複雑になり，MS同士の依存性やルーティングなどの管理が困難になるという問題がある。

これらの問題を解決する手段として，東芝は，東芝IoT基盤サービスHABANEROTSの開発において，MSAを採用するとともに，サービスメッシュ技術を導入している。ここでは，サービスメッシュ技術の導入によってMSAの問題をどのように解決できるかを述べるとともに，活用事例として，HABANEROTSの開発における，サービスメッシュの持つ可観測性を活用した手法について述べる。

2. HABANEROTSアーキテクチャー

2.1 HABANEROTSの概要

HABANEROTSは，CPSサービスを実現する上で必要となる共通的な機能を東芝グループ内のCPSサービス開発者にWeb API（Application Programming Interface）として提供することを目的としたWebサービスである。2020年6月時点で，HABANEROTSは，計65個のHTTPS（Hypertext

Transfer Protocol Secure）ベースのWeb APIを提供している。以下に，2.2節でも説明する，代表的なAPIの概要を示す。

⑴ Device API　デバイス情報の登録や，取得，更新，削除などを行う。

⑵ Timeseries API　各デバイスから収集されるセンサーデータなどの時系列データについて，登録や取得などを行う。

⑶ Role API　ユーザーごとのAPI実行権限や参照可能データを管理するためのロール情報の登録や，取得，更新，削除などを行う。

2.2 MSAの導入

HABANEROTSは，開発の迅速化や，可用性，スケーラビリティーといったサービス品質の向上を図るため，MSAを導入している。デバイス情報やロール情報といった，APIで扱うデータ種別を基にMSの分割を行っており，認可処理などのAPI共通処理とリクエスト内容に応じたMSの呼び分けを担うPublic-API MSを加えた計15個のMSで構成される。各APIの機能は，それらのMSを組み合わせて処理することで実現している（**図1**）。

MSを組み合わせた処理について，Timeseries APIを例に取り，**図2**で説明する。Timeseries APIにリクエスト（①）があると，Public-API MSで認可処理（②）が行われた後に，Role MSに対して当該APIを実行可能な権限が与えられているかをチェックする要求（③）が行われる。チェックを通過（④）した場合は，Public-API MSからDevice MSに対して，時系列データの取得対象となるデバイス情報の取得要求（⑤）が行われ，取得されたデバイス情報（⑥）を基に，Public-API MSからTimeseries MSに対して，時系列データの取得要求（⑦）が行われる。

このように，役割ごとにMSを分割して疎結合にしておき，MS同士が相互に通信して一つの機能やサービスを提供することで，MSの更新に伴う影響の極小化やMS単位での柔軟なスケーリングなどにつなげられ，可用性やスケーラビリティーの向上を図れる。また，それらの特性を生かすため，MSAはコンテナオーケストレーション技術と組み合わせて実現されることが多く，HABANEROTSもコンテナオーケストレーションツールのデファクトスタンダードであるKubernetes™を採用している。

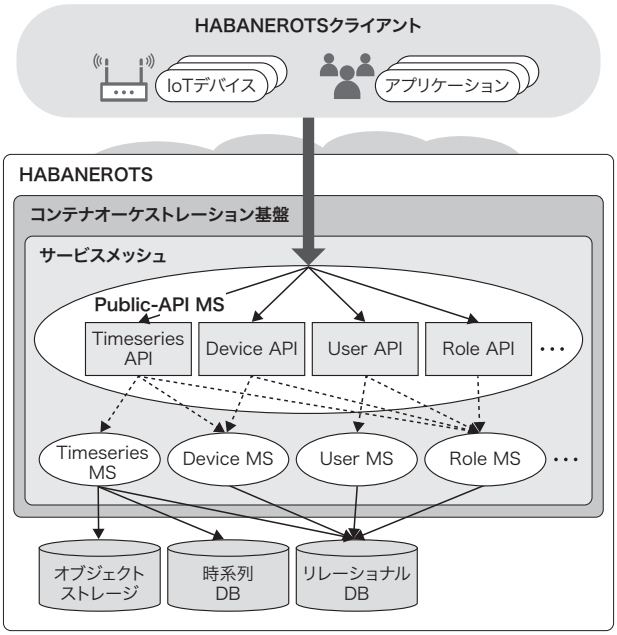
2.3 MSAが抱える問題

MSAには，2.2節で述べたメリットがある一方，以下のような問題がある。

⑴ コンテナなどの仮想的なインフラを用いる場合，MSの接続先情報（IP（Internet Protocol）アドレスやポート番号）の動的な変化に追従する必要がある。

⑵ 冗長化やスケーリングでのMS数の動的な変化に追従する必要がある。

⑶ MS数の増加に伴い，MS間通信の状態（依存関係や，エラー発生状況，処理時間など）の把握が難しくなる。



DB：データベース

図1. HABANEROTSのアーキテクチャー

開発の迅速化や，可用性，スケーラビリティーなどのサービス品質向上のため，MSAを導入することで，MS単位での冗長化や負荷分散が可能になる。

Architecture of HABANEROTS

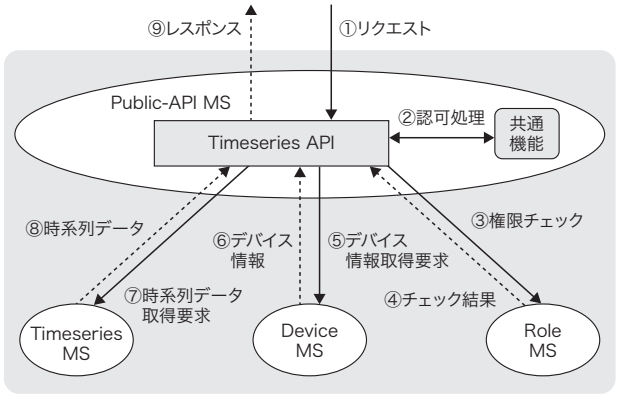


図2. MSAにおける処理シーケンスの例

APIで扱うデータ種別を基にMSの分割を行っており，複数のMSが相互に通信することで機能を提供する。高機能になるほど，MS間通信が複雑化しやすい。

Example of sequence of communications among microservices in MSA

⑷　MS数の増加に伴い，MS間の通信ルール（セキュリティーや，タイムアウト，リトライなど）の統制や，一元的なルール適用，調整などが難しくなる。

これらの問題を解決可能なライブラリーやフレームワークも存在しているが，MSごとに開発体制や実装言語が異なる場合があり，全てのMSについてそのような統制を図ることは容易ではない。また，MSごとにそうした仕組みを組み込む必要が生じるため，開発コストの増加を招く。

3. サービスメッシュ導入によるMSAの問題解決

2.3節で述べた，MSAが抱える問題を解決できる技術としてサービスメッシュがある。サービスメッシュでは，2.3節で挙げた問題に対応する機能を集約したプロキシーを用意し，各MSはそのプロキシーを介して相互に通信を行う。

Kubernetes™上でサービスメッシュを実現するツールとしては，Istio™[(1)]が代表的であり，HABANEROTSでもIstio™を採用している。Kubernetes™では，複数のコンテナをまとめたPodと呼ばれる単位で抽象化してコンテナの管理を行っており，Istio™は，その仕組みを生かしたサイドカーパターンを取っている（**図3**）。サイドカーパターンとは，Pod内にアプリケーションコンテナと独立した別コンテナ（サイドカー）を設け，アプリケーション実装を変更せずに新機能を持たせるデザインパターンである。Istio™は，サイドカーとして以下の機能を備えるプロキシーを，構成要素として持っている。

⑴　トラフィック管理
⑵　セキュリティー
⑶　可観測性

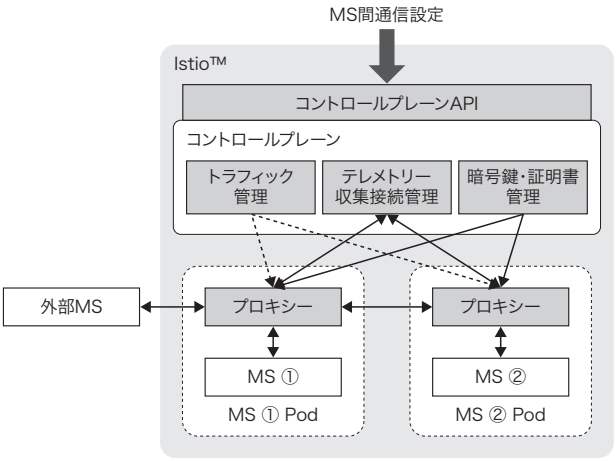


図3. Istio™アーキテクチャーの概要
MS間通信に関わる様々な機能をプロキシーが担い，MS同士は直接通信せず，プロキシーを介して相互に通信を行う。
Architecture of Istio™ service mesh

トラフィック管理に関する機能は，通信先となるMSにおいて，動的な変化への追従，リトライやタイムアウトなどの統一的な通信ルールの適用を可能にする。セキュリティーに関する機能では，mTLS（mutual Transport Layer Security）を用いることで，MS間通信の暗号化や相互認証を容易に実現可能であり，MSごとにそうした機能を組み込むための開発コストを削減できる。可観測性に関する機能は，2.3節で述べたようなMS間の通信状況の把握に役立つ機能であり，MS間のリクエストを追跡（トレーシング）する分散トレーシングと呼ばれる仕組みによって可能としている。

一方でサービスメッシュの導入は，システム構成が複雑化したり，ルーティング処理におけるオーバーヘッドがあることなどから，運用負荷の増大や性能低下を招いたりするおそれがある。小規模なシステムにおいてはそれらの影響がより顕著になるため，サービスメッシュの導入要否はシステムごとに検討が必要である。

4. サービスメッシュを活用した開発と継続的改善

HABANEROTSの開発では，3章で述べたIstio™のトラフィック管理と可観測性に関する機能を活用している。特に，分散トレーシングは各MSの開発，テストや運用，保守の効率化に役立つ機能であり，HABANEROTSでIstio™を採用する理由の一つとなっている。

4.1 HABANEROTSにおける分散トレーシングの活用

HABANEROTSの開発では，Istio™の分散トレーシング機能を，主に以下のようなシーンで活用している。

⑴　デバッグや障害調査時のエラー発生箇所の特定
⑵　性能テスト時のボトルネック調査
⑶　MS間の通信シーケンスの可視化
⑷　MS間の依存関係の可視化
⑸　APIごとの利用量集計

⑴は，開発フェーズと運用フェーズのどちらでも活用可能である。Istio™の分散トレーシングでは，一つのリクエストに対して一意なリクエストID（識別情報）が割り振られる。同一リクエストの処理で通信を行ったMSのログには同じリクエストIDが記録されるため，同じリクエストIDを持つログを抽出することで，特定のリクエストについて全てのMS間通信の処理結果を容易に確認できる。つまり，どのMSからどのMSに対するどの処理でエラーになっているかの特定が容易になるため，効率的な調査が可能となる。

同様に，分散トレーシングのログは，⑵にも活用できる。ログには，リクエストIDや処理結果のほかに，各MSでの処理に要した時間も記録されているため，リクエストIDで抽出されたログの中から処理時間が大きくなっている箇所を

見つけることで，容易にボトルネックを特定することが可能になる。これについては，4.2節で詳しい実例を説明する。

⑶や⑷は，サードパーティーツールを組み合わせることで容易に実現可能である。

更に⑸は，トレーシングログをログ分析基盤に蓄積しておくことで，ログに含まれるドメインや，リクエストパス，リクエストメソッドといった情報からどのAPIがどの程度使われているかを集計したり，リクエストの応答時間から応答性能を，リクエストの応答ステータスからエラー発生頻度を集計したりできる。そして，これらの集計結果を基にすることで，今後の機能強化や不具合改修といった継続的改善につなげられる。

4.2　分散トレーシングを活用したボトルネック調査

HABANEROTSの開発における実例として，Timeseries APIの性能テストでのボトルネック調査について詳しく説明する。Timeseries APIでは，図2のようなシーケンスでMS間通信が行われている。分散トレーシングのログから各MSでの処理時間を確認したところ，Timeseries MSでの処理時間が長くなっていることが判明した。Timeseries MSの主要な処理は，時系列DB（データベース）からのデータ取得であり，ボトルネックとなりやすい箇所であるため，そこが要因である可能性が最初に疑われた。しかし，Timeseries MSから時系列DBに対する通信の処理時間は数ms程度しか掛かっておらず，Timeseries MS内において，時系列DBからのデータ取得処理以外の処理に原因があることを特定できた。そこで，Timeseries MSのソースコードを調査した結果，Timeseries MSでのキャッシュ処理のロジックに問題があり，正しくキャッシュできずに不要なデータ参照処理が繰り返し行われていたことが性能低下の真因と特定された。この真因特定に至るまでに要した時間は約1時間で，大半がTimeseries MSのソースコード調査に要した時間である。

一方で，分散トレーシングを活用していない場合，ボトルネック箇所の特定方法は，以下の2通りが考えられる。

⑴　全MSのソースコードを網羅的に調査

⑵　全MSのソースコードに処理開始／終了時刻のログ出力を埋め込んだデバッグ用モジュールを作成して再現試験を実施

⑴において，仮に一つのMSのソースコード調査に1時間掛かるとした場合，Timeseries APIには四つのMSが関係するため，調査には計4時間掛かることになる。また，⑵のログ出力の埋め込みにMS一つ当たり15分掛かるとすると，四つのMS全てにログ出力を埋め込むと計1時間掛かる計算となる。したがって，再現試験に1時間掛かると仮定し，ボトルネック箇所特定後のソースコード調査に1時間掛かるとすると計3時間となる。このように，分散トレーシングを活用した場合と比較すると，真因特定までに，⑴の方法では4倍，⑵の方法では3倍の時間が必要となる。

⑵の方法に関しては，あらかじめ全てのMSでログ出力が行われるように実装しておけば，調査に際してのデバッグ用モジュールの作成時間を省略できるが，実装時に同程度の工数が掛かることになる。分散トレーシングを活用すると，このようなビジネス機能とは直接関係しない処理の実装に掛かる工数を削減できるため，ビジネス機能の実装に開発リソースを集中でき，開発の俊敏性向上につなげられる。

5.　あとがき

MSAが抱える問題の解決手段として，サービスメッシュの導入を挙げ，HABANEROTSの開発におけるサービスメッシュの活用事例について述べた。

今後も，CPSサービスのように迅速な開発サイクルが求められるシステムに有効な手法を開発し，システム開発の効率化や継続的改善を行っていく。

文　献

⑴　Istio community. "Istio". Istio. <https://istio.io>, (accessed 2020-06-10).

内田　正之　UCHIDA Masayuki
デジタルイノベーションテクノロジーセンター
技術開発部
Digital Innovation Technology Development Dept.

樋口　靖和　HIGUCHI Yasukazu
デジタルイノベーションテクノロジーセンター
技術開発部
Digital Innovation Technology Development Dept.