細粒度マルチテナント化フレームワークによる高密度SaaS基盤の実現

島村 栄・副島 賢司
黒田 貴之・西村 祥治

要　旨

複数のテナントのアプリケーションを収容するSaaS型クラウドを実現する上で、インフラ当たりのテナント収容密度がサービス提供コストに大きく影響します。
本稿ではテナント間でアプリケーションを共有して、個別の処理の単位でテナント分離を行うことにより、テナント収容密度を極限まで引き上げる細粒度APフロー制御技術について紹介します。

キーワード

●SaaS ●マルチテナント ●Aspect Oriented Programming

1. まえがき

近年、ネットワークを介してサーバ上のソフトウェアの機能を提供するSaaS（Software as a Service）と呼ばれるサービスが広がりを見せています。SaaSは、利用者がハードウェアやソフトウェアを所持することなく、利用量に応じた料金でサービスとして提供されたソフトウェアの機能を使用できる点が特徴で、クラウドコンピューティングの一形態ととらえることができます。

SaaSにはさまざまなユースケースが考えられますが、SaaS事業者が企業や団体を対象にしたサービスを提供する場合、このサービスの利用者のことをテナントと呼びます。単にサービス利用者と区別するのは、テナント自体がエンドユーザを抱えているケースがあるためです。例えば、SaaS事業者が顧客管理サービスを提供する場合、このサービスを利用して自らの顧客情報を管理する企業がテナントに当たります。

一般に中小企業向けのSaaS事業においては、大企業向けサービスと比較してより低いコストでサービスを提供することが求められます。サービス提供コスト低減のためには、同一のハードウェアやミドルウェアをテナント間で共有するマルチテナント化技術が必須です。本稿では、テナント間でアプリケーションのプロセスまでを共有し、個別の処理の単位でテナント分離を行うことにより、ハードウェア当たりのテナント収容数を極限まで高めることを可能とする細粒度APフロー制御技術について紹介します。

2. マルチテナント化方式

2.1 マルチインスタンス型

アプリケーションをSaaS形態で運用するためにマルチテナント化する手法には、いくつかの類型があります。最も単純には、テナントごとにサーバハードウェア、または、仮想マシンを用意し、それぞれのOS上でアプリケーションを稼働させることによって容易にマルチテナント環境を構築することができます（**図1** のa及びb）。

インフラの利用効率を高めるには、より上位の層までテナント間で共有する必要があります。例えば、OSまでをテナント共通にしてプロセスをテナントごとに用意する方法、アプ

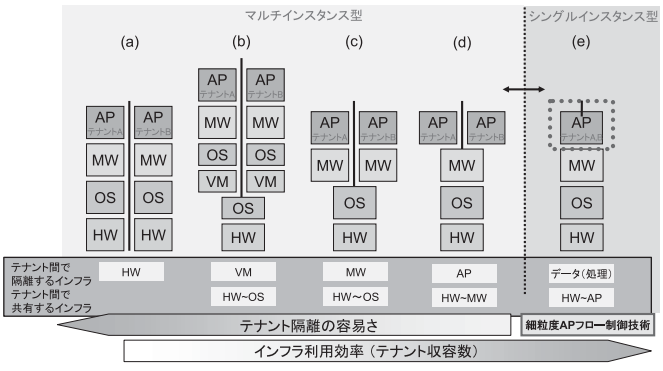

図1　マルチテナント化の類型

リケーションサーバまでをテナント共通にしてテナントごとにアプリケーションプログラムを配備する方法などが考えられます（図1のc及びd）。

これらの方法は、テナントごとの動作状況のモニタリング、制御が比較的容易であることからサービスレベルを維持しやすく、マルチテナント化に当たってアプリケーションプログラムの変更を伴わないというメリットがあります。この一方で、テナントと同数のアプリケーションプログラムを配備するため、配備するだけで収容するテナント数に比例した量のハードウェアやOSのリソースを消費します。

これらのマルチテナント化はアプリケーションプログラムを複数配備、稼働させる方法であることから、本研究ではマルチインスタンス型のマルチテナント化と呼びます。

2.2 シングルインスタンス型

ハードウェア当たりのテナント収容効率を更に向上させるために、アプリケーションプログラムまでをテナント共通にして、処理の単位でテナントを分ける方法があります（図1のe）。この方法ではアプリケーションプログラムをマルチテナント化のために変更する必要が生じることと、各テナントの処理が相互に影響を及ぼしやすいというデメリットがある半面、収容するテナント数にかかわらずアプリケーションプログラムを1つだけ配備するため、ハードウェア当たりのテナント収容効率を極限まで向上可能というメリットがあります。

本研究ではこのマルチテナント化方式をシングルインスタンス型のマルチテナント化と呼びます。本稿では特に断りなくマルチテナント化と書いた場合、シングルインスタンス型を指すものとします。

3. TIP-Throughモデル

この章では、シングルインスタンス型のマルチテナント化を実現するTIP-Throughモデルについて説明します。

3.1 概要

本研究では、細粒度APフロー制御技術の核として、Tenant ID Pass Through（TIP-Through）モデルを考案しました。TIP-Throughモデルとは、処理の対象となるテナントの識別子（テナントID）を伝播させる仕組みと、テナントごとに異なる処理が必要となった時点で適切な処理に割り振る仕組みをアプリケーションのフローと独立に定義し、後付けでアプリケーションに適用するというモデルです。

このようにTIP-Throughモデルでアプリケーションの処理に必要な機能とマルチテナント化に必要な機能を分離することにより、下記の効果が得られます。

- 既存アプリケーションのマルチテナント化の効率化
- 個別SIで使用するシングルテナントアプリケーションとSaaS向けのマルチテナントアプリケーションのコードベース共通化

3.2 TIP-Throughモデルの構成

TIP-Throughモデルは、(1) 識別、(2) トンネル、(3) 割り振りの3つの要素から構成されます。**図2**にTIP-Throughモデルの概念図を示します。以下、それぞれの機能、動作例について説明します。

(1) 識別

識別はアプリケーションの処理開始時にアプリケーション処理に割り込んで、その処理がどのテナント向けのものか特定する機能です。

例えば、HTTPによるリクエストURLのPathの先頭にテナントを識別するための文字列を挿入するとした場合、“http://testservice.example.com/tenant1/”へのアクセスに対しては“tenant1”という文字列をキーにどのテナント向けの処理かを識別し、該当アクセスに対する処理を特定するためのIDと組にしてトンネル機能に記録します。

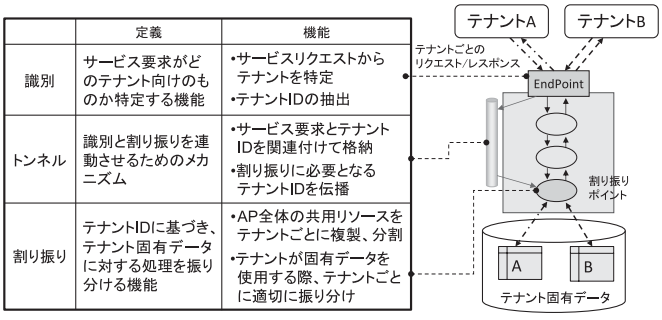

図2　TIP-Throughモデル概念図

(2) トンネル

トンネルは、識別と割り振りを連動させるためのメカニズムです。識別機能でアプリケーション処理とテナントIDを関連付けて格納し、割り振り箇所までテナントIDを伝播し、そこからの参照を可能にする機能です。

例えばJava EEサーバで動作するアプリケーションであれば、アプリケーション処理を特定するスレッドIDとテナントIDの組をトンネル機能に格納します。通常のメソッドコールが続く限りスレッドIDをキーとして問い合わせればその処理に該当するテナントIDを取得できます。ただし、RMIやSOAPなどによるリモート呼び出しやスレッド間通信などでスレッドをまたがるケースでは呼び出し元、呼び出し先で個別にトンネル機能を生成し、トンネル間でテナントIDを連携させる仕組みが別途必要です。

(3) 割り振り

割り振り機能はアプリケーション処理に割り込んで、テナント固有のリソースに対する処理を振り分ける機能です。アプリケーション全体の共用リソースを複製・分割してテナントに割り当てたり、データベースなどのテナント固有データをテナントごとに適切に切り替えたりする機能です。この部分がアプリケーションの処理フローをテナントに応じて制御する部分となります。

例えば、多くのWebアプリケーションでは処理に必要なデータをDBから検索、格納しますが、マルチテナント化する場合、テナントごとのデータが混ざらないように適切に格納する必要があります。このようなテナント固有リソースに対する処理の手前で、トンネル機能に現在処理中のテナントIDを問い合わせ、そのテナントIDに該当するテナント用のデータベースへアクセスするよう処理フローを制御することによって、マルチテナント化を実現します。

4. TIP-Throughモデルのアプリケーションへの適用

TIP-Throughモデルのアプリケーションへの適用方法としてさまざまなものが考えられます。主な適用方法として、(a) 直接ソースコードに埋め込む、(b) アプリケーションサーバやアプリケーションフレームワークのプラグイン機構などを使用する、(c) Aspect Oriented Programming（AOP：アスペクト指向プログラミング）を使用する、などが考えられます。

本研究では（c）のAOPの技術を応用したTIP-Throughモデルの適用基盤を、J2EEアプリケーション向けのマルチテナントライブラリとして試作しています。以降、本章ではAOPの概説とマルチテナントライブラリの実装について説明します。

4.1 Aspect Oriented Programming（AOP）

AOPとは、オブジェクト指向プログラミングで扱いづらい多くのクラスにまたがる共通な特徴を分離して記述するプログラミング手法です。例えば、ログ出力や例外処理といったクラスを横断する共通的な機能をアプリケーション本来の処理とは分離して記述し、アプリケーション処理の中に埋め込むことができます。この記述をアスペクトと呼び、AOP実装が提供するコンパイラやローダが、アスペクトをアプリケーションコードの指定した位置に割り込ませます。

本研究では、マルチテナント化のための特徴をアスペクトとして記述し、AOPの持つ機能によってアプリケーションに適用します。具体的には、TIP-Throughモデルの識別、割り振り機能をアスペクトとして記述し、AOPによってアプリケーション処理に割り込ませます。AOPによる適用のメリットを以下に挙げます。

- アプリケーション側のコードの修正、再コンパイルが必ずしも必要でない
- 設定のみでマルチテナント化に伴う追加コードの適用、解除が可能であるため、マルチテナント版、シングルテナント版でコードの多くの部分を共用化でき、切り替えも容易

4.2 マルチテナントライブラリ

本研究では、アプリケーションに対するTIP-ThroughのAOPによる適用を容易にするマルチテナントライブラリを試作しました。以降、マルチテナントライブラリの主な構成要素について**図3**に沿って説明します。

(1) 部品ライブラリ

TIP-Throughの識別、トンネル、割り振り機能は適用対象となるアプリケーションの作りや、達成したいマルチテナントアプリケーションの仕様に応じていくつかの類型にパターン分けすることができます。この類型に応じた部品を用意して、識別や割り振り機能の実装工数を削減すること

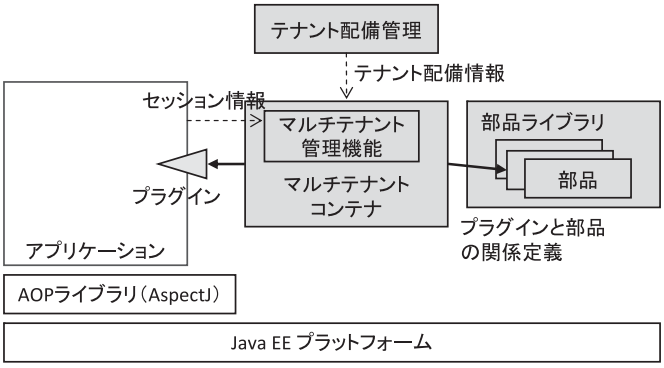

図3　マルチテナントライブラリの概要構成

により、マルチテナント化のコストを大幅に下げることができます。

(2) マルチテナントコンテナ機能

本機能は下記で説明する各管理機能に識別、トンネル、割り振りの各部品がアクセスするための入り口となる機能です。マルチテナントコンテナ機能は、マルチテナントライブラリを構成する各管理機能を統合して連動させるコンテナとしての役割を持ちます。

(3) テナント配備管理機能

本機能はサービスに対する新しいテナント追加を行う機能です。テナント配備管理機能は、マルチテナント化されたアプリケーションに対して、テナント固有のリソース情報のマルチテナント管理機能への登録、及びAPサーバやDBサーバなどへのリソースの生成、登録を行います。

(4) マルチテナント管理機能

本機能はマルチテナントライブラリの中心であり、主に下記の機能から構成されます。

・ セッション管理機能

セッション管理機能は、1つのトンネルではテナントIDの伝搬を実現できない場合に、トンネル間でテナントIDの連携を実現する機能です。例えば、一連のHTTPリクエストに対する複数のアプリケーション処理を特定のテナントに対するひとまとまりの処理として識別する際に使用されます。

・ プラグイン管理機能

本機能は、部品ライブラリに定義された識別、割り振りの機能部品に対し、テナント固有の情報を適用した上でその機能部品を実行する機能です。テナント固有情報は、トンネルからテナントIDを取得し、後述のテナント管理機能から該当テナントの固有情報を取得します。

・ テナント管理機能

テナント配備管理機能から設定される、テナントごとのアクセスURLやデータベース名、リソースファイルのパスなどのテナント固有情報を管理する機能です。

4.3 マルチテナント化手順

マルチテナント化はすべて自動でできるわけではなく、アプリケーションのコードを見ながらTIP-Throughモデルの各機能を適用していく必要があります。特に既存アプリケーションをマルチテナント化する際にad-hocに適用すると、重大なデグレードを引き起こす可能性があります。

そこで本研究ではマルチテナントライブラリを用いることを前提とした既存アプリケーションのマルチテナント化手順を示したガイドを作成しました。このガイドは下記の構成で必要な手順を説明しています。

・ 設計：割り振りの対象となるリソースへのアクセス箇所の特定、識別方法の決定
・ 実装：割り振りや識別機能コードの実装
・ 配備：アプリケーションやミドルウェアに対して上記実装を適用するための設定
・ テスト：デグレードとマルチテナント化のテスト

5. 考察

本研究の細粒度APフロー制御技術によって簡単な在庫管理アプリケーションをマルチテナント化したところ、1サーバで50テナント程度までは特に動作に問題が発生することなく配備することができました。各テナントへのアクセス頻度にも依存しますが、更に高密度なテナント収容も実現可能です。比較のため、図1のdで示したマルチインスタンス型でのマルチテナント化モデルで配備したところ、20テナント程度でOSのファイルハンドルを使い尽くして動作に支障をきたす状態となりました。このことから細粒度APフロー制御技術ではシステムリソースを効率的に利用しているといえます。

また、本技術でアプリケーションロジックからマルチテナント化要件を分離することで、既存アプリケーションのマルチテナント化の工数削減に効果があると考えています。

一方、課題としては現状のマルチテナントライブラリでは

同じアプリケーションを多数のテナントで共有することが前提となっており、テナントごとのカスタマイズ要件に対応していません。しかし、カスタマイズ要件に対してもTIP-Throughモデルを応用して特定の処理をテナントごとに切り替えることによって、ある程度対応可能であることが実証できており、マルチテナント化ライブラリへの統合も可能だと考えています。

また、特定のテナントに非常に高い負荷がかかった場合、ほかのテナントの処理性能に影響が及ぶという点があります。これはシングルインスタンス型のマルチテナント化では現状避けられない課題で、テナントごとの性能モニタリングとリソース制御に関する技術開発が必要です。

更に、本技術が多くのリソースをテナント間で共有するモデルである点からも、ほかのテナントのリソースと取り違える可能性がないかなどのセキュリティ面での検討も必要です。

6. むすび

本稿ではハードウェア当たりの収容テナント数を極限まで向上することが可能なシングルインスタンス型のマルチテナント化について説明し、これを実現するための細粒度APフロー制御技術について説明しました。この細粒度APフロー制御技術の核であるTIP-Throughモデルに沿ったマルチテナント化により、既存アプリケーションのマルチテナント化を効率的に実現することが見込めます。

*OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及び その他の国における登録商標です。

執筆者プロフィール

島村 栄
サービスプラットフォーム研究所
主任研究員
情報処理学会会員

副島 賢司
サービスプラットフォーム研究所
主任

黒田 貴之
サービスプラットフォーム研究所
情報処理学会会員

西村 祥治
サービスプラットフォーム研究所
主任
情報処理学会会員