D-011

マイクロサービスアーキテクチャ向け分散トランザクション管理技術の提案と評価

Proposal and Evaluation of Distributed Transaction Management Technology for Microservice Architecture

野田 昌太郎† 大越 淳平† 樫山 俊彦† 馬場 恒彦†
Shotaro Noda Jumpei Okoshi Toshihiko Kashiyama Tsunehiko Baba

1. はじめに

近年，スケーラビリティやアジリティの向上を目的に，独立してデプロイ可能なサービスの集合体としてアプリケーションを構築するマイクロサービスアーキテクチャ（以下，MSA）が注目されている。MSA においては，サービスごとにデータベースを保持する設計方式が一般的であるため[1]，2 つのサービスのマッシュアップにより口座間送金サービスを実現する場合など，分散トランザクションによりサービス間のデータの整合性を維持する必要がある。

MSA 採用時の分散トランザクション手法として Saga パターン（以下，Saga）[1]が広く知られている。Saga では，各サービスのトランザクション（以下，ローカルトランザクション）を一連のワークフローとして管理，及び実行することで，分散トランザクションを実現する。一方で，Saga を用いた分散トランザクションは複雑化しやすいという課題がある。例えば，Viggiato らによるアプリケーション開発者に対するサーベイ調査[2]によると，分散トランザクションの複雑さが MSA における主要な課題の一つとして挙げられている。具体的な複雑さとしては，以下の（1）Saga 自体の実装，および（2）システム障害によるデータ不整合への対応の 2 つが考えられる。

（1）**Saga 自体の実装**：前述の通り，Saga では，複数のローカルトランザクションをワークフローとして管理する必要がある。また，ローカルトランザクションの実行失敗時には，補償トランザクション[1]によりこれまで実行した処理を打ち消す必要がある。これらワークフローの状態管理やローカルトランザクションの実行制御により Saga 自体の実装が複雑化する。

（2）**システム障害によるデータ不整合への対応**： Saga は複数のサービスに跨って実行されるため，各サービスの障害や通信障害などによって処理が途中で停止し，サービス間でデータの不整合が発生する場合がある。アプリケーション開発者は，各システム障害やそれによって生じるデータの不整合を想定した上でサービスを実装する必要があるが，システム障害は多岐に渡り Saga の実装を複雑化させる要因となる。

以上の状況を鑑み，本研究では（1）の複雑さを低減する Saga 実装支援機能，及び（2）の複雑さを低減するシステム障害とそれによって生じるデータ不整合の対処方法の体系を備えた分散トランザクション管理フレームワークを提案する。

2. 分散トランザクション管理フレームワーク

2.1 Saga 実装支援機能

本研究で提案する Saga の実装支援機能を図 1 に示す。なお，Saga の実装形態としてコレオグラフィ型とオーケストレータ型の 2 つが存在するが，本稿では利用頻度の高さ[3]の観点からオーケストレータ型を想定した 。本支援機能は，（1）の複雑さがワークフローの状態管理やローカルトランザクションの実行制御に起因することに着眼し，これらを実現する機能群をライブラリとして分離することを特徴とする。具体的には，Saga のワークフローの定義を BPMN（Business Process Modeling Notation）[4]として読み込み，有向非巡回グラフ（Directed Acyclic Graph，DAG）として状態を管理する。また，DAG の状態に応じて実行すべきタスク（各参加サービスのローカルトランザクションに対応）の実行順序を制御する。ユーザはライブラリが提供する API（Application Programming Interface）越しにタスクの取得，及び実行結果の登録を実施する。これにより，ユーザは各タスクの内部処理（具体的には各参加サービスへの API 呼び出しによるローカルトランザクションの実行など）の実装に注力することが可能となる。

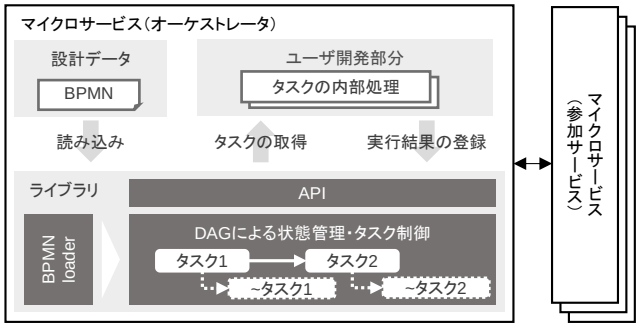

図 1 Saga 実装支援機能

2.2 データ不整合と対処用デザインパターンの体系

前述の通り，Saga を実装する際には，Saga 実行中のシステム障害によって発生するデータの不整合への対処方法を検討する必要がある。そこで，本研究では，Saga 実行中に発生するシステム障害に対し，生じるデータ不整合，及び対処方法となるデザインパターンを整理することで体系化を実施した（表 1）。

マイクロサービスに関連する既存のデザインパターンの体系として Richardson による体系[1]が挙げられる。本研究においても当該体系を基に対応を取り（例えば#3），不足しているデザインパターン（#4-7）については新たに Retry パターン，及び Idempotent publisher/consumer パターンをそれぞれ追加した。Retry パターンは，何らかの理由により（典型的にはネットワーク障害によるリクエストの消失）メッセージの再送が必要になった際にその再送を行うデザインパターンである。また，Idempotent publisher /consumer パターンは，リクエストした処理の多重実行の防止を目的に，リクエストに付与する一意な ID（冪等 ID と呼ぶ）を

† 株式会社 日立製作所 研究開発グループ Hitachi, Ltd. Research & Development Group

[1] これまで実行したローカルトランザクションの打ち消し処理に相当する特別なローカルトランザクション

発行，及び管理するデザインパターンであり，Retry パターンと併用することで参加サービスから応答が得られない場合のデータ不整合に対処することが可能となる

3. 評価

評価として，2.1 節で述べた Saga 実装支援機能によるアプリケーション開発者の開発工数の削減効果を定量的に評価した。具体的には，図 2 に示すシステム構成で複数銀行間の資金移動処理を実施するマイクロサービス（資金移動サービス）を対象に，開発工数に大きなが影響のあるコード量にて評価した。なお，銀行 A， B に対応する残高管理サービスは Java の開発フレームワークである Quarkus[5]を用いて実装し，資金移動サービスは Quarkus と Saga 実装支援機能を併用して実装した。

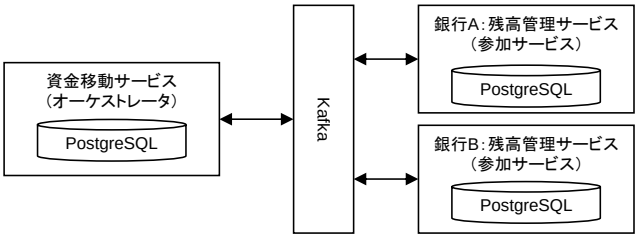

図 2 評価環境

Saga 実装支援機能によるコード量の削減効果の評価結果を図 3 に示す。ライブラリを利用しないスクラッチ実装の場合は，ライブラリと同等のコード量で実装する必要があると仮定し，資金移動サービス実装時に削減されたコード量を測定した。この結果，実装支援機能を利用しない場合と比較し，提案する Saga 実装支援機能によりコード量の 73.6%を削減可能と分かった。

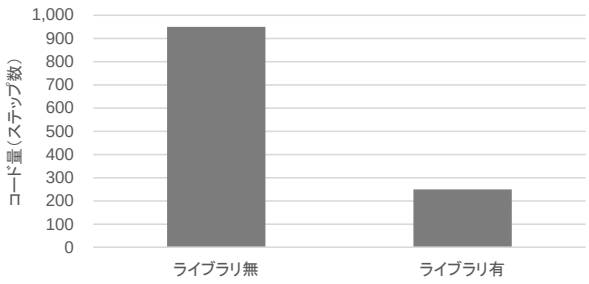

図 3 評価結果

4. おわりに

MSA における分散トランザクションの複雑さを低減するため，Saga の実装支援機能，及びシステム障害により発生するデータ不整合と対処用のデザインパターンの体系を備えた分散トランザクション管理フレームワークを提案した。本フレームワークによりコード量を 73.6%削減可能となり，MSA 採用時のアジリティ低下の要因となっていた Saga 実装時の複雑さの大幅な低減が可能となった。今後の課題としては，表 1 で未定義となっているデザインパターンの定義，Retry などの対処用デザインパターンの実装支援機能の開発，および開発工数低減効果の評価が挙げられる。

参考文献

[1] C. Richardson, “Microservice Architecture,” https://microservices.io/. [Accessed: 2021/3/19].
[2] M. Viggiato, R. Terra, H. Rocha, E. Figueiredo, M. T. Valente, “Microservices in Practice: A Survey Study,” Arxiv, 2019.
[3] L. Rodrigo, Z. Yongluan, V. S. A. Marcos, L. Yijian, and K. Marcos, “Data Management in Microservices: State of the Practice, Challenges, and Research Directions,” Arxiv, 2021.
[4] Object Management Group, “Business Process Model and Notation,” https://www.omg.org/spec/BPMN/2.0/PDF. [Accessed: 2021/3/17].
[5] Quarkus, https://quarkus.io/, [Accessed: 2021/6/2].

表 1 システム障害と発生するデータ不整合に対応したパターン体系（N/A：未定義）

#	システム障害	生じるデータ不整合	デザインパターン
1	参加サービスの応答を待つ間にオーケストレータのサービス障害が発生	参加サービスの処理状態（実行完了）とオーケストレータで管理される処理状態（リクエスト済かつ未完了）が一致しない	*N/A*
2	参加サービスの応答を受け取ってから処理状態を永続化する間にオーケストレータのサービス障害が発生		
3	あるタスクの処理状態を永続化してから，次のタスクと対応するリクエストを送信する間にオーケストレータのサービス障害が発生	オーケストレータの処理状態（リクエスト未発行，リクエスト済かつ未完了）が不明となり，オーケストレータの障害前の処理状態と一致しない	・Transactional messaging [1]
4	オーケストレータの送信したリクエストが参加サービスに届く前に消失	参加サービスの処理状態（未実行，実行完了，実行失敗など）が不明で，オーケストレータが管理する自身の処理状態と一致しない	・Retry ・Idempotent publisher/consumer
5	参加サービスが送信した応答がオーケストレータに届く前に消失		
6	受け取ったリクエストの処理中に参加サービスにサービス障害が発生		
7	オーケストレータが参加サービスから処理失敗の応答を受け取った	オーケストレータの期待する処理結果（実行完了）と参加サービスの処理結果（実行失敗）が一致しない	・Retry