D-007

分散トランザクション管理ミドルウェアによるレガシーシステムの管理方式
Management Method for Legacy Systems using Distributed Transaction Management Middleware

坂井 秀行† 今木 常之†
Hideyuki Sakai Tsuneyuki Imaki

1. はじめに

デジタルトランスフォーメーションの潮流に伴いレガシーシステムのモダナイズが進んでおり、複数のマイクロサービス[1]のリソース更新からなる分散トランザクションの管理が必要となっている。多様な分散トランザクション管理ミドルウェアが提供される中、分散合意判定機能により基幹系業務に適用できるような耐障害性をもつものもあるが、管理対象の機能をレガシー言語である COBOL から Java(登録商標)に作り変える必要があるケースもあり、移行コストが高いという課題につながる。そこで本研究では上記のようなミドルウェアにおいて、COBOL 資産を維持したままトランザクション管理を行うことで移行コストを低減するための構成を提案する。

2. 分散トランザクション管理ミドルウェア

本研究で検討対象とする分散トランザクション管理方式である Two-Phase Commit(2 相コミット)[2]および分散トランザクション管理ミドルウェア[3]について説明する。

2.1 Two-Phase Commit (図 1)

クライアントアプリケーションは複数のデータベースに対して更新処理を行い、トランザクションコーディネータにそれらの更新処理のコミット(決着)を要求する。 トランザクションコーディネータは各データベースにプリペア処理を要求する。各データベースはコミットの準備が整っていれば OK を返し、整っていなければ NG を返す。トランザクションコーディネータは全てのデータベースから OK が返ってきたら各データベースにコミット処理を要求する。もし 1 つでも NG が返ってきたら各データベースにロールバック処理を要求する。この手続きにより、全てのデータベースを更新した状態か、全てのデータベースを更新しない状態のどちらかを取ることができるようになる。

2.2 分散合意判定機能をもつミドルウェア (図 2)

図 2 に示す、オーケストレータ、インターセプタ、メディエータ、エンティティサービス、パーティシパントをもつ分散トランザクション管理ミドルウェアを考える。ここで、メディエータは図 1 におけるトランザクションコーディネータに相当し、多重化されているものとする。また、各コンポーネントは Java で構築されているものとする。

オーケストレータはクライアントアプリケーションからインターセプタ経由で業務リクエストを受信し、業務処理を行う。そして、各エンティティサービスに向けた業務リクエストを送信する。各エンティティサービスは業務処理を行い、対応するパーティシパントにデータベースの更新処理を依頼する。各パーティシパントは対応するデータベースに更新処理を行い、処理結果をエンティティサービス経由でオーケストレータへと返す。インターセプタは処理結果を受信すると、各メディエータにトランザクションの決着を指示する。各メディエータは各パーティシパントにプリペア処理を指示する。パーティシパントは対応するデータベースにプリペア処理を行い、処理結果(OK/NG)を各メディエータに返す。各メディエータは受信した処理結果を元にコミットまたはロールバックの判定を行い、各パーティシパントに送信する。各パーティシパントは受信した判定結果を元にコミット処理またはロールバック処理を対応するデータベースに行う。

以上のフローにより Two-Phase Commit 方式の分散トランザクション管理を実現する。このとき、メディエータが多重化されていることで、一部のメディエータに障害が起こっても判定を行えるようになり、可用性が向上する。

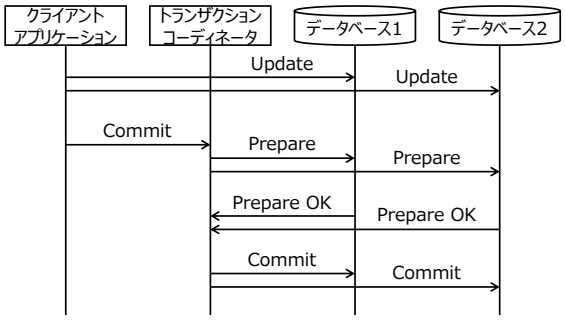

図 1 Two-Phase Commit

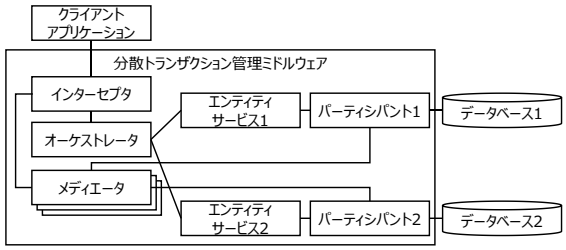

図 2 分散合意判定機能をもつミドルウェア

3. レガシーシステム

本研究で検討対象とするレガシーシステムについて説明する。ここで、レガシーシステムとは、1980 年代から 1990 年代に COBOL 等のレガシー言語を用いて構築されたシステムのこととする。現在でもミッションクリティカルな基幹系システムで利用されている場合もあり、デジタルトランスフォーメーションの対象となっている。

3.1 モノリシックレガシーシステム (図 3)

モノリシックなレガシーシステムの例を図 3 に示す。オーケストレータ、機能 A、機能 B が主要なコンポーネントであり、それぞれ COBOL で開発されたものとする。オーケストレータはクライアントアプリケーションから業務リ

†株式会社日立製作所 Hitachi, Ltd.

クエストを受信すると、業務処理を行うとともに、機能 A および機能 B に個別の業務リクエストを送信する。機能 A および機能 B は業務処理を行うとともに、SQL 実行部を経由してデータベース A およびデータベース B に更新処理を行う。各機能で呼び出される SQL は COBOL コード中に埋め込まれた埋め込み SQL である。オーケストレータは各機能から処理結果を受信すると、トランザクションコーディネータを通じて Two-Phase Commit を行う。

ここで、業務処理部には複雑なロジックが含まれている場合もあり、レガシー資産と呼ぶこととする。そのレガシー資産は COBOL で開発されていることから、そのままでは、2.2 節で説明したミドルウェアを適用できない。

4. レガシー資産を活用する方式の提案

モノリシックなレガシーシステムをデジタルトランスフォーメーションの一環としてマイクロサービス化し、2.2 節で説明したミドルウェアによって分散トランザクションを管理することを考える。このとき、動作実績のあるレガシー資産は引き続き活用する。

4.1 アーキテクチャ (図 4)

図 3 のレガシーシステムのオーケストレータ、機能 A そして機能 B をマイクロサービス化し、2.2 節の分散トランザクション管理ミドルウェアを適用したアーキテクチャを図 4 に示す。

オーケストレータマイクロサービスでは、COBOL で開発されたオーケストレータの業務処理部を活用する。また、REST API 経由で業務リクエストを受信したらオーケストレータを呼び出す機能を設ける。次に、機能 A マイクロサービスおよび機能 B マイクロサービスと通信するための REST 通信機能を設ける。さらに、Java から COBOL への接続、およびその逆の COBOL から Java への接続を仲介する C 言語で開発された接続機能を設ける。そして、オーケストレータには各マイクロサービスへの REST 通信機能の呼び出し部を設ける。

機能 A マイクロサービスでは、COBOL で開発された機能 A の業務処理部を活用する。また、REST API 経由で業務リクエストを受信したら機能 A を呼び出す機能を設ける。次に、パーティシパントと通信するための JDBC(登録商標)通信機能を設ける。さらに、Java から COBOL への接続、およびその逆の COBOL から Java への接続を仲介する C 言語で開発された接続機能を設ける。そして、機能 A には JDBC 通信機能の呼び出し部を設ける。ここで、レガシーシステムでは埋め込み SQL であった SQL コマンドを、JDBC 通信機能に渡すための変数の形式に置き換える。機能 B マイクロサービスも同様の構成を取る。

以上のアーキテクチャにより、COBOL 資産である業務処理部を活用しながら、Java のサービスを管理対象としているミドルウェアによる分散トランザクション管理を可能とし、全体として動作実績の活用と高可用性を得られる。

4.2 実装方式

提案アーキテクチャにおける追加コンポーネント、すなわちオーケストレータ呼び出し機能(Java)、機能 A および機能 B 呼び出し機能(Java)、REST 通信機能(Java)、JDBC 通信機能(Java)、Java から COBOL への接続機能(C 言語)、

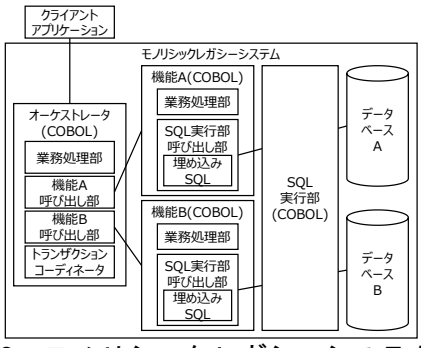

図 3　モノリシックレガシーシステム

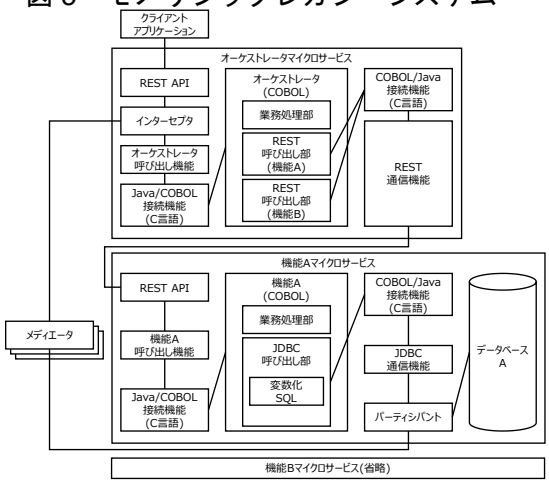

図 4　提案アーキテクチャ

COBOL から Java への接続機能(C 言語)、REST 呼び出し部(COBOL)、そして JDBC 呼び出し部(COBOL)には、業務処理が含まれていないため、必要な変数値の設定以外は自動実装が可能である。

例えば、REST 通信機能は Java の RestTemplate 等のライブラリの利用、JDBC 通信機能は JDBC ドライバの利用を行う標準的なプログラムである。また、Java から COBOL への接続機能と、COBOL から Java への接続機能は、JNI(Java Native Interface)[4]という仕様に従った C 言語のプログラムで実装が可能であり、基本的には Java から C 言語を経由して COBOL へ、またはその逆へ、関数のパラメタを引き渡す方法が主な実装内容となる。

5. おわりに

本研究では、COBOL 等のレガシー言語で開発された業務機能を活用した形のマイクロサービスを、高可用性を実現する分散トランザクション管理ミドルウェアで管理するためのアーキテクチャを提案した。

現在、当該アーキテクチャのコンポーネントを実装し、動作検証を進めているところである。

参考文献

[1] Sam Newman, “マイクロサービスアーキテクチャ”, O’REILLY (2016).
[2] Martin Fowler, “Two-Phase Commit”, https://martinfowler.com/articles/patterns-of-distributed-systems/two-phase-commit.html
[3] 西谷 淳平, “分散合意を用いたクラウドネイティブトランザクション Paxos Commit”, CloudNative Days Tokyo 2022 (2022).
[4] Oracle, “Java Native Interface 仕様”, https://docs.oracle.com/javase/jp/1.5.0/guide/jni/spec/jniTOC.html