分散サーバー上でのリアルタイム・トランザクション処理

長妻 令子　大崎 博靖

Real -Time Transaction Processing between Distribution Servers in Some Regions

Reiko Nagatsuma and Hiroyasu Ohsaki

各地に分散している疎結合のサーバーをまたがるトランザクションを，リアルタイムで同期を取って処理することは容易ではない．本論文では，分散サーバー間のサービスを柔軟に連携して，リアルタイム・トランザクション処理を実現するための課題と解決案を検討する．その結果，呼び出し元を変更することなく，呼び出し先のプロトコルを柔軟に切り替える「振り分け機能」が連携のオーバーヘッドが少なく，リアルタイム・トランザクション処理を実現する現実的な解決案となることを示す．

In this paper, we describe the characteristics and problems concerning the design and development of applications with a focus on calls of services on distributed servers. Such application structures are required when an application is built by SOA. When a distributed service calls another service on a remote server, it is necessary to conceal the difference of method on each service call. Furthermore, calls of multiple services must be treated as a single transaction. A detailed discussion is held concerning an assigning function applied at the design phase as a solution to the problems and the usage of WS-Atomic Transaction for maintaining the atomicity of a transaction.

Key Words & Phrases：SOA, Webサービス, EJB, 分散トランザクション, Web Services Atomic Transaction
SOA, web service, EJB, distributed transaction, Web Services Atomic Transaction

1．はじめに

多国間をまたがる受発注システムのように疎結合システム間で即時に同期を取る必要のあるシステムにはリアルタイム・トランザクション処理が必要である．しかし，複数のアプリケーションサーバー間にまたがる複数のリソースの整合性を保つための技術が成熟していなかったため，従来はキューなどを使用していったんトランザクションを終了させ，失敗した場合にデータを元に戻すといった処理を補償プロセスとして別途記述していた．このため，コーディングやメンテナンスのコストが発生していた．近年，複数の技術分野で分散トランザクションの仕様が提唱され，これまでは難しいとされてきた分散サーバー間での2-Phase Commit（2相コミット：以下2PC）の仕組みが成熟してきている．分散サーバーで2PCを用いることができれば，障害回復はトランザクションマネージャーに任せられるので，リアルタイム・トランザクションが可能になる．

また，呼び出す先によって柔軟に呼び出し方法を変更したいという要件を実現する手法についての課題があった．Enterprise Service Bus（以下ESB）のようなアプリケーション統合の基盤となる複数プロトコルサポートのメッセージバスを適応して解決することも可能であるが，リアルタイムに同期処理を必要とするアプリケーションに対して既存のESB製品の導入を考えたとき，データ変換によるレスポンスタイムの遅れや，密結合のトランザクションが選べないといった課題がある．これの課題を解決しシームレスに呼び出しを行う機能が必要である．ここではそれを「振り分け機能」と呼ぶ．なお密結合のトランザクションについては，2章で詳しく述べる．

本論文では，SOA（Service Oriented Architecture）の概念を取り入れ，SOAで規定する「サービス」（以下サービス）の間でリアルタイム・トランザクションを行う疎結合なシステムの構築について述べる．

以下2章で分散トランザクション処理の要件・技術およびその課題について説明する．3章では課題点をクリアするために適応した手法としての振り分け機能について述べる．実装事例を4章に記述し，5章ではこれらの適応した分散トランザクション処理や振り分け機能についての有効性を考察する．

提出日：2006年8月31日　　再提出日：2007年9月28日

なお, 分散サーバー間において発生するトランザクション処理を「分散トランザクション処理」と定義し, 特に記述がなければ分散サーバーは疎結合なサーバー連携とする.

2. トランザクションについて

2.1 トランザクション処理の要件

トランザクション処理には次の2種類がある.

(1)通常密結合なシステムで採用されることの多いACIDトランザクション[1]

(2)通常疎結合のシステムで採用されることが多いロング・ランニング・トランザクション

(1)と(2)の違いを説明するために, 簡単なトランザクションの例をあげる.

サーバーA上のサービスを呼び出してDBを更新した後に, サーバーB上のサービスを呼び出し, サーバーB上のDBの更新が成功したらすべてが成功する, というアプリケーションを構築する場合を考える.

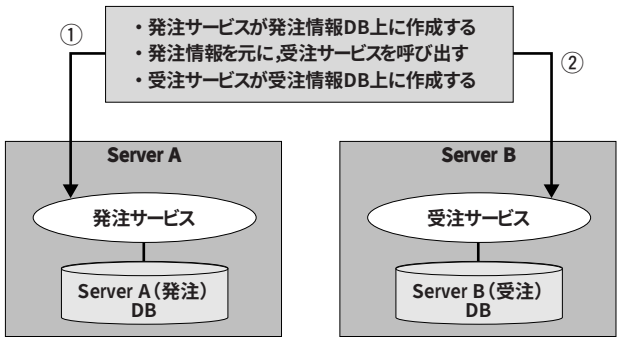

図1. トランザクションの例

図1において, ①の処理の際に発生したトランザクションを②に引き継いで, ②の処理が成功したら②をコミットし, ①もコミットする, というアプリケーションの場合は, 前者のACIDトランザクションを採用する. この場合, ダーティリードを防ぐために, トランザクション処理が終了するまでレコードがロックされるので, 長時間のトランザクション処理には向いていない.

②の処理時間自体が長くなるなどリソースのロックを行いたくないアプリケーションの場合は, 後者のロング・ランニング・トランザクションを採用する. このトランザクション処理を採用した場合は, いったん①の呼び出し部分がコミットされているので, ②の処理が失敗した場合は, ①の処理のキャンセル処理を別途記述してそれを呼び出す, という仕組みが必要となる.

どちらのトランザクションを採択するのかは業務要件によって異なる. 例えば図1中の①で発注側が100個発注した場合, ②の受注側でも, 発注が確定したタイミングと同じタイミングで100個受注したという情報を作成することが業務要件として必要であれば, (1)のACIDトランザクションを採択する.

このACID特性を実現するために, 分散サーバー間では2PCが利用される. 2PCとは, 分散システム内でトランザクションに関わっているすべての対象の処理が問題ない場合のみ, トランザクションを完結させることができるプロトコルである.

2PCを実現するために策定された仕様にはX/Openというコンソーシアムが策定したDTP(Distributed Transaction Processing)や, オブジェクト指向の分散トランザクション仕様として策定されたOMG CORBA OTS[2]などがある. ここでは今後SOAとして採用する可能性が高いもののうち, 分散サーバー間でもACIDトランザクションが実現できる2つの標準について述べる. 1つはJava™ Transaction API [1](以下JTA), もう1つはWeb Services Atomic Transaction [5](以下WS-AT)である. JTAはJava Transaction Services(以下JTS)の実装であり, ほかのJTS実装(広い意味でのOTS実装)で記述されたアプリケーション間では分散トランザクション処理が行える. しかし, OTS以外の呼び出しには対応していない. ただしオブジェクトのまま通信を行うのでパフォーマンスは良い. 一方WS-ATはWebサービス以外のトランザクションを使用する場合も2PCが可能であるが, 通信のためにSimple Object Access Protocol(以下SOAP)の形式に変換しているのでパフォーマンスに難点がある. 以下JTAとWS-ATを詳しく述べる.

2.1.1 JTAを用いたトランザクション

Java 2 Enterprise Edition(J2EE)はトランザクション・サービスの仕様であるJTSとプログラミング・インターフェースの仕様であるJTAを定義している. これらはCORBA OTSがベースで, 2PCプロトコルによる分散トランザクション処理を標準でサポートしている.

JTAは, 複数のコンポーネントやリソースマネージャ間にまたがることができるトランザクションで処理を実行するためのAPIであり, 複雑なトランザクション処

1 ACIDトランザクションとは, 次の4つの特性の頭文字を取ってこのように呼ばれている.
- A:Atomicity(原子性―処理の単位が分割不可であること)
- C:Consistency(一貫性―処理の順番の影響がないこと)
- I:Isolation(隔離性―ほかのトランザクションの影響を受けないこと)
- D:Durability(耐久性―トランザクションの完結後にデータの変更がなされない)

2 OMG : Object Management Group
CORBA : Common Object Request Broker Architecture
OTS Object Transaction Service

理を開発することが可能である. Enterprise Java Bean（以下EJB）はJTAのインターフェースを使用してトランザクション管理を実現しているが, EJBを使用して開発する場合は, トランザクション管理を開発者がコード内に実装するよりも, EJBコンテナーにトランザクション処理を任せた方が, 開発効率が良い［2］.

2.1.2 WS-Atomic Transaction

IBM, Microsoft®, BEA Systemsが提唱するWebサービスの仕様（Web Services Transaction［3］, 以下WS-Transaction）が公開され, 各ベンダーではこの仕様にしたがって, 分散トランザクション処理を行うアプリケーションの開発を行うことができるようになった. それまでは, 標準の仕様が存在せず, オープンオペラビリティの保証ができなくなるため, トランザクションが複数サーバーに分散する異なるアプリケーション間を伝達できるように実装することができなかった. このWS-Transactionには, 以下の3つが含まれる.

- Web Service Coordination［4］：サービス連携で使用するコンテキストの作成方法に関する仕様
- Web Services Atomic Transaction（以下WS-AT［5］）：ACIDトランザクションを実現するための仕様
- Web Services Business Activity［6］：ロング・ランニング・トランザクションを実現するための仕様

それまでは分散トランザクションの実行をサポートするWebサービスの仕様は存在しなかったためトランザクションを伝達することが不可能であったが, WS-Transactionを用いることによりWebサービスでもトランザクション管理が可能となった.

このようにJTAはJTA同士での2PCが実行され, 一方WS-TransactionはJTA-WS-AT間のトランザクション処理でも, それが1トランザクションとして動作する場合, 2PCが実行される.

2.2 分散トランザクション処理を実現するための課題

図2に, 分散サーバー上の典型的な処理例を示す.

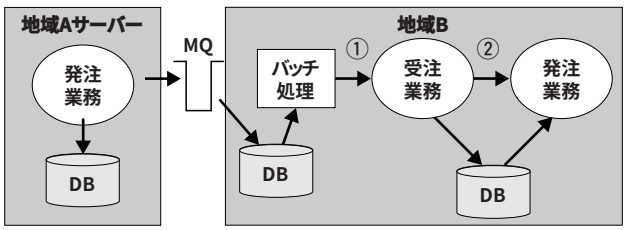

図2. 分散サーバー上の典型的な処理

図2で地域Aのサーバーからの発注情報が入ったキューは, 地域Bでいったん DBに書き出される. バッチが定期的にDBからデータを取得して（図2中の①の線）, 地域Bの受注のリクエストを作成する. 受注が確定するために必要な在庫がない場合, 受注を受けたのとは別のタイミングで発注業務を呼び出す必要がある（図2の②の線）. このように従来は, 地域Aと地域Bではタイムラグがあり, リアルタイム処理が実現していなかった.

このような疎結合サーバー間でデータの整合性を取るためには, 障害回復のための補償プロセスを記述する必要があり, 後々のメンテナンスが複雑になり, アプリケーション障害の原因となることが多い.

また, 図2では地域Aから地域Bの呼び出し方法をアプリケーション側が意識する必要があるので, アプリケーション側の開発コストがかかる.

これら2つの課題の解決手段について以下で述べる.

3. リアルタイム処理の実現手法

3.1 サービス間の連携方針

サービス間連携を行うため, 分散サーバーのリアルタイム・トランザクション処理, および, サービス間のシームレスな呼び出しに着目する.

図2における発注業務や受注業務をサービスとしてとらえ, SOA的なアプローチで呼び出すことを考える. これらのサービスがサーバー間でどのような連携を行えばよいのか, 複数のトランザクション連携パターンを検討する. このとき, 以下の2点を考慮する必要がある.

（1）サービスからサービスを呼び出す処理の同期を取るため, ACIDトランザクションを採用し, 2PCを実現させる.
（2）呼び出す相手によって異なるトランザクションを意識することなく呼び出せる.

3.2 リアルタイム・トランザクション処理

リアルタイム・トランザクションを実現するためには障害回復が必要である. 分散サーバー間でも2PCを行えるトランザクションを使用すれば, 失敗した場合はトランザクションマネージャーがロールバックを行うので, この問題は解決できる.

分散トランザクション処理で2PCを行うことができるJTA（JavaによるEJB呼び出し）かWS-AT（Webサービス呼び出し）を採用し, 分散サーバーのリアルタイム・トランザクションを実現する. EJB呼び出しの場合は, オブジェクト呼び出しであるので, Webサービスのように SOAP-HTTPで送るための形式に変換する必要がなく, レスポンスタイムは比較的短くてすむ. 逆にWebサービスの場合は, いったん変換する必要があるが, 相手がJavaでなくてもWebサービスの定義があれば呼び出せるもの（.NETなど）であればよいという利点がある.

これらを考慮して, 同一のサーバー内呼び出しの場合はパフォーマンスを考慮してEJBとし, 別サーバーのサービスを呼び出す場合は, オープンオペラビリティを保証するため, Webサービスを使用するというように, 相手先によって呼び出し方法を切り替えることを考える.

3.3 振り分け機能

図2に記述した発注業務/受注業務をSOAでいうところのサービスとし, 呼び出し先の実装に依存しない形式だとしてサービスの連携を考える. 相手先によって呼び出しの実装を変えるとロジックが複雑になるからである.

このような疎結合システムのサービス連携に対するアプローチとしてESBというアーキテクチャーの概念を導入する. ESBはSOA実現のための機能として考えられており, 複数の呼び出し方法をサポートするメッセージバスである. ESBベンダーは複数の高度な技術を統合させてこれらの概念を実装したミドルウェア製品として提供している.

実際にアプリケーションに適応できるESB製品には下記の課題がある.

- ESB製品として提供されている従来の製品の多くは, サービス振り分けの機能にコーディングが必要であり, 余分な実装コストがかかる. またデータの変換ロジックが入り, パフォーマンスに懸念がある. 呼び出しパターンとして内部サーバー内の呼び出しが多い場合このような製品を使用するのは得策ではない.
- 従来のメッセージングハブの機能を持つ製品はMQを間に介し, ロング・ランニング・トランザクションしか保障されていない.

既存の製品では解決できなかったこれらの課題をクリアするために, ESBの考え方を用いて, 呼び出し先によって異なる分散トランザクション処理の違いを吸収する必要がある.

このように呼び出し先によって動的に呼び出しのプロトコルを切り替える機能をここでは「振り分け機能」と呼ぶことにする. 振り分け機能は, 要求されたサービスが, どのサーバーにいて, どの呼び出し方法を使用すればよいのかを判断して, 別のサービスを呼び出す機能を持つ.

この振り分け機能は, 従来のESB製品のようにいったんWebサービスの形にして再び別の呼び出し方法に変換するという形式ではなく, EJB/Webサービスを直接呼び出す機能である. そのため, 呼び出し元に, EJB/Webサービスで共通のインターフェースを提供し, データ変換ロジックが少なくなるような構成にした. このことにより, 変換が間に入ることによるパフォーマンスの低下を防ぐことができる.

具体的には,
(1)呼び出しデータオブジェクト(以下DTO)
(2)呼び出し共通インターフェース(以下Invokerインターフェース)
(3)オブジェクト取得機能(以下Locator)
の3つを定義する.

DTOは, 呼び元/呼び先情報及び送信情報を定義し, LocatorではDTOに記述された情報よりどのInvokerを使用するのかを取得する機能を備える.

Invokerインターフェースは下記のようになる.

```
interface Invoker{
        Object invoke();
        void translation();
}
```

Webサービス/EJBはこのインターフェースを実装し, 各々の呼び出し方法を記述する.

サービスは下記のようにしてLocatorを呼び出して呼び出しを実装したクラスを取得する.

```
Interface invokerIF = Locator.getInvoker(dto)
```

このように, サービスは振り分け機能を使用して呼び出し方法を取得する. このときに複数の呼び出し方法が1つのトランザクション処理として引き継がれる必要がある. 例えば, 同一サーバー上にあるサービスAがサービスBを呼び出す場合, まずサービスAでEJBコンテナーのトランザクションが開始され, サービスBを呼び出す際にトランザクションはJTAによって管理され引き継がれる. 次に別のサーバーのサービスCが呼び出される場合はWebサービスで呼び出される. 呼び出されたWebサービスはWS-ATの機能によりトランザクション管理を行っており, 2章で述べたとおりサービスBのトランザクションを引き継ぐことが可能である. このような振り分け機能を追加することにより複数のサービスを複数の呼び出し手法で連携しても2PCが実現され, サービス側はそのことを意識することがないという構造にできる.

4. 実装事例

4.1 アプリケーション概要

筆者らは複数の地域に販売管理の拠点を持ち, その拠点毎に異なる販売管理アプリケーションが動いているというシステム全体を再構築する機会を得た. 統一したアプリケーションとして再構築し, 複数の地域間でリアルタイムな分散トランザクション処理を保障することが要件であった. アプリケーションの概要は下記のとおりである.

(1)受注・発注・出荷・着荷など8つの業務を持つ販

売管理アプリケーションである.

(2)EJBパターン[7]をアプリケーション基盤のデザインとして採用する.

(3)日本にある既存のホストに対して要求を出す機能は残す.

(4)全世界で共通な業務ロジック部(Coreロジック部)と地域独自のロジック部に分けて開発を行う.

(5)地域Aと地域Bにおいてリアルタイムな分散トランザクション処理が必要となる

このアプリケーションのレイヤーの関連図を**図3**に示す.

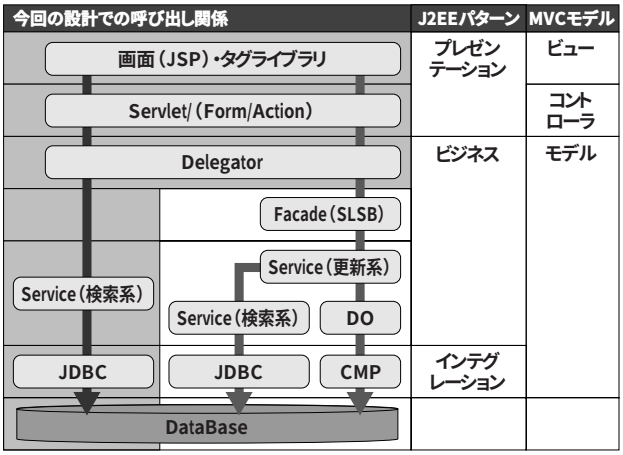

図3. アプリケーションの階層構造

このような階層構造にすることにより, 各レイヤーの責務をクリアにし, ビジネスロジックの集中化を行った. 次に, アプリケーションの構造がSOA的な考え方に沿っているかどうかを検討した.

今回構築した販売管理アプリケーションは, 中に受注・発注・出荷・着荷などの業務を持つ. これらの業務1つにつき1つ, 業務の流れをコントロールするクラスを定義した. これはビジネスロジックをほとんど持たずに, フローのみを持つような形式とした. つまり, 業務として意味のある最小単位であるコンポーネントをどのような順番で呼び出すのかという流れのみをコントロールするようなクラスである. これがSOAでいうサービスに相当する. これらのサービスの単位がビジネスユースケースの単位となっており, サービスの単位として適当である[8].

4.2 サービスとコンポーネントの関係について

サービスの下の階層であるコンポーネントの呼び出し関係においては, 下記のような注意点がある. **図4**で示すように受注サービスが, 受注要求を実行した後に, 不足在庫を発注するというシナリオを考える. 発注サービスが呼び出すべき発注要求というコンポーネントを受注サービスからも呼び出している設計をすると発注要求に変更が起きた場合に, 受注サービスも発注サービスも変更する必要が出てきて, 自立性が失われる. 正しくは**図5**に示すように, サービスとコンポーネントの対応付けがないコンポーネントを呼び出す場合には, 対応しているサービスを呼び出すように設計する必要がある. (図5の発注サービスと受注サービス間の矢印部分)

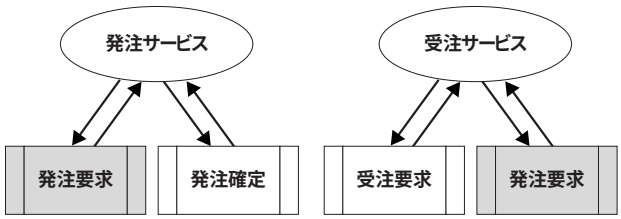

図4. 複数サービスから同一コンポーネントの呼び出し

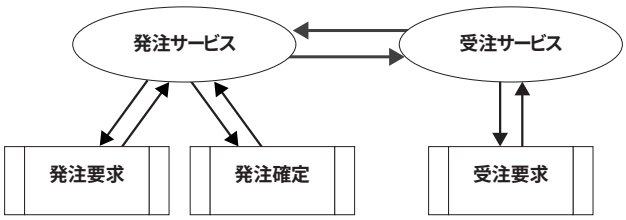

図5. 変更後のサービス呼び出し

4.3 各地域固有の処理について

サービスの呼び出し口を変更しないためには地域特有の処理(帳票出力や税金計算など)を除いて, 各地域に共通のロジックのみを抽出する必要がある. 各地域固有で行うべき処理を開発する際は, 各サービスが呼び出すコンポーネント内の処理を地域向けに変更する. ただしサービス間をつなぐインターフェースは変更しないという大前提を守ることができるように共通部分のロジックを抽出することが重要である.

4.4 サービス間の連携方針

全地域が一斉にこのアプリケーションに置き換わるわけではなく, 各地域に順次展開していく方針であるため既存のホストを経由する場合が存在する. 既存の処理では, 地域のアプリケーションがMQに情報を書き出し, 受け取ったホスト側でバッチ処理を行った後, 関連する別の地域のサーバーに情報を送信し, サーバー間の情報のやり取りを行っていた. このMQ呼び出しの実装も行う. MQの処理の部分はリアルタイム処理ではなくバッチ処理である.

図6に, 振り分け機能を含んだ地域間の連携図を示す.

振り分け機能は, 要求されたサービスが, どのサーバーにいて, どの呼び出し方法を使用すればよいのかを判断して, 別のサービスを呼び出す機能を持つ. 図6の例で言えば, 地域Aの発注のサービスは, 受注

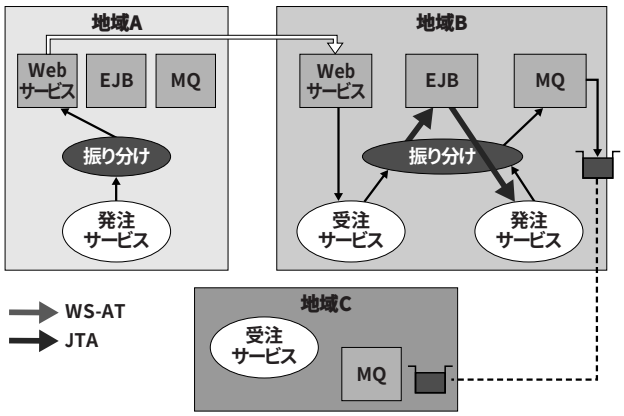

図6.　振り分け機能を含んだ連携図

サービスを呼び出す必要があるが, 受注サービスがどのサーバーにあり, どの呼び出し方法で呼べばよいのか発注サービスが知る必要はなく, 振り分け機能が, 発注サービスの持つ供給者情報から, 地域Bの受注であることを判断し, 地域Bは地域Aとは別のサーバーであるから, Webサービスで呼び出すことが必要であることを判断する. さらに地域Bの受注サービスが, 在庫不足で発注を起こす場合も同様に, 振り分け機能は受注情報から同一地域に発注を出すことを判断し, 同一地域であるので, EJB呼び出しを行えばよいことを判断する.

5. 考察

このように, WebサービスおよびEJB/MQを使用してリモートに配置されているサービス間の連携を行う販売管理アプリケーションの開発・テストを行った.

5.1　パフォーマンスについての考察

開発したアプリケーションはまだ本番環境には導入されていないため, テスト用サーバーにてパフォーマンスを計測した. マシンのスペックは,

- OS：Windows® 2000
- CPU：2.0GHz
- Memory：2GB

ネットワークは

- 100BASE-TX with Switching HUB

を使用した.

テストにはEJBとWebサービスで同一のアプリケーションを使用した. テスターなど自動でパフォーマンスを計るツールは使用せず, 別のサーバーのサービスを呼び出してから戻ってくるまでの時間をログ上から取得して記録した. **表1**に, アプリケーションサーバーが起動した後最初にサービス間呼び出しを行った際にかかった時間と, 2回目以降にかかった時間の平均をEJBとWebサービスそれぞれに対して計測した結果

表1.　サーバー間呼び出しパフォーマンス

	1回目	2〜5回目平均
EJB	5.1sec	0.477sec
Webサービス	15.9sec	0.572sec

を示す.

Webサービス呼び出しの1回目とそれ以降の違いが顕著であることがわかった. Webサービス呼び出しのコードはキャッシュする構造にはなっていないため, 相手先を探しに行く部分による違いではないと思われる. WebSphere® Application Server v6.0上に出力されるログを調べたところ, 1回目の呼び出しだけSOAPを解析するための初期化処理を行っているログが出ていた. この処理に14秒近くかかっており, 1回目に時間がかかる原因はこの初期化処理にあることがわかった.

EJB呼び出しでもWebサービス呼び出しでも1回目には時間がかかることが判明した. この問題を解決するためには, 例えば, アプリケーションサーバーの起動時に, 他サーバーのサービスの呼び出しを行うといった処理が必要である.

次に, EJBとWebサービスの呼び出しの速度の違いについてであるが, 2回目以降は0.1sec程度の差であった. テストに使用したアプリケーションは, 図1のように複数のサーバーにそれぞれに配置されているDBを更新する必要があり, このDBへのアクセスが遅いため全体としては1sec程度の時間がかかっている.

1secに対して0.1secであれば気にならない程度であるが, ESB製品のように, すべてWebサービスとして呼び出すような構造である場合, ESB製品の内部でさらに別プロトコルに変換して呼び出すという流れとなるので, 0.1secほどの違いが積み重なり, パフォーマンスが落ちることが予測される. しかし, 振り分け機能により, 呼び出しのプロトコルを呼び出し元から変えているので, このようなパフォーマンスのロスは起こらない構造となっている.

5.2　展開の現状

現在, この販売管理アプリケーションはプロトタイプ版として中国にて稼動しており、その他の地域においては, 今後地域要件をカスタマイズしてアプリケーションを開発し, 随時サービスインしていく予定である.

地域開発のカスタマイズをする場合はサービス呼び出し部分の変更はしないという前提である. 呼び出し口の変更により呼び出せる地域と呼び出せない地域が出てくる混乱を避けるためである. 今後, 各地域によってサービス呼び出しをカスタマイズする要件が出てきた場合に呼び出し部分をどのようにしていくかは今後の課題である.

6. おわりに

本論文では新しい分散トランザクション技術であるWS-ATおよびJTAを用いたトランザクション処理を使用することによって, それまで実現できていなかった分散サーバー間のリアルタイム・トランザクション処理を容易に行うことができることを示した. また, 分散サーバー間をシームレスに呼び出すための連携方法やそのトランザクション管理といった問題に対しての現実的なアプローチを提案した.

これらのことから, 分散トランザクション処理にWebサービスやEJBを用いることの可能性を示し, またESB的なアプローチである振り分け機能を考えることにより, その呼び出し方法の違いを吸収できる構造を示した.

このようにSOA的なアプローチによって, 複数の地域で異なる要件を持ち, リアルタイムな分散トランザクション処理を必要とするアプリケーション構築の指針を示した.

謝辞

本論文の執筆にあたっては, 大和事業所の一柳 晶子さんに助言およびレビューいただきました. この場をお借りして, お礼申し上げます.

参考文献

[1] Java Transaction API (JTA)
http://java.sun.com/products/jta/ (2001).
[2] 日本アイ・ビー・エム システムズエンジニアリング: WebSphere V5.0 開発者必携ガイド1 J2EE入門, 技術評論社 (2003).
[3] Web Services Transaction
http://www-128.ibm.com/developerworks/library/specification/ws-tx/#atom
[4] Web Services Coordination (Specification)
http://www-06.ibm.com/jp/developerworks/webservices/021122/j_ws-coor.pdf (2005).
[5] Web Services Atomic Transaction (Specification)
http://specs.xmlsoap.org/ws/2004/10/wsat/ (2005).
[6] Web Services Business Activity Framework,
http://specs.xmlsoap.org/ws/2004/10/wsat/ (2005).
[7] フロイド マリネスキュー: EJBデザインパターン, 日経BP社 (2003).
[8] ディルク・クラフツィック他: SOA大全 サービス指向アーキテクチャ導入・設計・構築の指針, 日経BP社 (2005).

日本アイ・ビー・エム株式会社
ソフトウェア開発研究所
WebSphereサービス
ソフトウェア開発エンジニア

長妻 令子 Reiko Nagatsuma

[プロフィール]

1996年日本IBM入社. 入社後, Web系のソフトウェア製品開発およびアプリケーション開発を担当.

金融機関のお客様のACM端末開発担当に携わり, 2004年よりある製造業のお客様の販売管理アプリケーション構築のフレームワーク設計／開発に従事している.

reiko@jp.ibm.com

日本アイ・ビー・エム株式会社
ソフトウェア開発研究所
WebSphereサービス
アドバイザリー・ソフトウェア開発エンジニア

大崎 博靖 Hiroyasu Ohsaki

[プロフィール]

1989年, 日本IBM入社. 以来, ワードプロセッサーなどのオフィス・アプリケーション製品や, ワークフロー製品の設計・開発に従事. 後に, ソフトウェア開発研究所にてビジネス・プロセス・マネージメント関連製品の開発およびWebSphere関連製品に対するサービスを担当している.

ohsaki@jp.ibm.com