BEA Tuxedo

BEA Tuxedo CORBA アプリケーションの スケーリング、分散、およびチューニング

BEA Tuxedo リリース 8.0
8.0 版
2001 年 10 月 31 日

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA Tuxedo CORBA アプリケーションのスケーリング、分散、およびチューニング

Document Edition	Date	Software Version
8.0	2001 年 10 月 31 日	BEA Tuxedo 8.0

目次

このマニュアルについて

対象読者 vii
e-docs Web サイト viii
マニュアルの印刷方法 viii
関連情報 viii
サポート情報 ix
表記上の規則 ix

1. BEA Tuxedo CORBA アプリケーションのスケーリング

BEA Tuxedo CORBA アプリケーションのスケーリングについて 1-2
アプリケーションのスケーラビリティ要件 1-2
BEA Tuxedo スケーラビリティの特徴 1-3
オブジェクト状態管理の使用 1-3
CORBA オブジェクト状態モデル 1-4
状態を持たないオブジェクトおよび状態を持つオブジェクトのインプリメント 1-5
パラレル・オブジェクト 1-8
サーバ・プロセスおよびサーバ・グループの複製 1-9
サーバ・プロセスおよびサーバ・グループの複製について 1-9
コンフィギュレーションのオプション 1-10
サーバ・プロセスの複製 1-11
サーバ・グループの複製 1-12
マルチスレッド・サーバの使用 1-13
マルチスレッド CORBA サーバについて 1-13
マルチスレッド CORBA サーバを使用すべき場合 1-14
コーディングの推奨事項 1-15
マルチスレッド CORBA サーバのコンフィギュレーション 1-15
ファクトリ・ベース・ルーティングの使用 (CORBA サーバのみ) 1-16
ファクトリ・ベース・ルーティングについて 1-16
ファクトリ・ベース・ルーティングの特性 1-17
ファクトリ・ベースのインプリメンテーションのしくみ 1-18

UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション....1-19
パラレル・オブジェクトの使用....1-19
パラレル・オブジェクトについて....1-20
パラレル・オブジェクトのコンフィギュレーション....1-23
受信するクライアント接続の多重化....1-24
IIOP リスナ / ハンドラ....1-24
ISH プロセス数の増加....1-25

2. CORBA サーバ・アプリケーションのスケーリング

Production サンプル・アプリケーションのスケーリングについて....2-2
設計上の目標....2-2
アプリケーションのスケーリング方法....2-2
OMG IDL の変更....2-4
状態を持たないオブジェクト・モデルの使用....2-4
サーバ・プロセスおよびサーバ・グループの複製によるスケーリング....2-5
Production アプリケーションでのサーバ・プロセスの複製....2-6
Production アプリケーションでのサーバ・グループの複製....2-8
Production アプリケーションの複製サーバ・プロセスおよびグループのコンフィギュレーション....2-9
ファクトリ・ベース・ルーティングによるスケーリング....2-11
Production アプリケーションでのファクトリ・ベース・ルーティングについて....2-11
UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション....2-12
ファクトリでのファクトリ・ベース・ルーティングのインプリメンテーション....2-15
実行時の処理....2-17
設計上の追加考慮事項....2-18
設計上の追加考慮事項について....2-18
Registrar オブジェクトおよび Teller オブジェクトのインスタンス化....2-19
学生の登録が確実に正しいサーバ・グループ内で行われるようにする....2-20
Teller オブジェクトが適切なサーバ・グループでインスタンス化されるようにする方法....2-22

アプリケーションをさらにスケーリングする方法 2-23

3. CORBA アプリケーションの分散

アプリケーションを分散する理由 3-2
アプリケーションの分散について 3-2
分散アプリケーションの利点 3-2
アプリケーション分散の特性 3-3
データ依存型ルーティングの使用 (BEA Tuxedo ATMI サーバのみ) 3-4
データ依存型ルーティングについて 3-4
データ依存型ルーティングの特性 3-5
分散アプリケーションの例 3-5
分散アプリケーションにおける UBBCONFIG セクションの例 3-6
UBBCONFIG ファイルのコンフィギュレーション 3-7
分散アプリケーションにおける UBBCONFIG ファイルについて 3-7
GROUPS セクションの変更 3-8
SERVICES セクションの変更 3-10
INTERFACES セクションの変更 3-12
ROUTING セクションの作成 3-13
factory_finder.ini のコンフィギュレーション (CORBA アプリケーションのみ) 3-14
ルーティングをサポートするためのドメイン・ゲートウェイ・コンフィギュレーション・ファイルの変更 3-15
ドメイン・ゲートウェイ・コンフィギュレーション・ファイルについて 3-15
DMCONFIG ファイルの DM_ROUTING セクションのパラメータ (BEA Tuxedo ATMI のみ) 3-16

4. CORBA アプリケーションのチューニング

アプリケーション資源の最大化 4-2
MSSQ セットを使用すべき場合 (BEA Tuxedo ATMI サーバのみ) 4-2
システム制御のロード・バランシングの有効化 4-4
複製されたサーバ・プロセスおよびグループのコンフィギュレーション 4-5
マルチスレッド・サーバのコンフィギュレーション 4-6
データベースの相互運用のための OPENINFO パラメータの設定 4-6
マルチスレッド・サーバのコンフィギュレーションに使用するパラメータ 4-7

インターフェイスへの優先順位の割り当て 4-8
サーバへのサービスのバンドル (BEA Tuxedo ATMI サーバのみ)....4-9
サービスのバンドルについて 4-9
サービスのバンドルが必要な場合....4-10
性能オプション 4-11
アプリケーション・パラメータによる効率の向上 4-12
MAXDISPATCHTHREADS....4-12
MINDISPATCHTHREADS 4-14
MAXACCESSERS、MAXOBJECTS、MAXSERVERS、MAXINTERFACES、および MAXSERVICES パラメータの設定 4-14
MAXGTT、MAXBUFTYPE、および MAXBUFSTYPE パラメータの設定 4-15
SANITYSCAN、BLOCKTIME、BBLQUERY、および DBBLWAIT パラメータの設定 4-15
アプリケーション・パラメータの設定 4-16
IPC 要件の決定....4-17
システム・トラフィックの測定....4-18
システム・トラフィックとボトルネックについて 4-19
システム・ボトルネックの検出例....4-19
UNIX におけるボトルネックの検出 4-20
Windows におけるボトルネックの検出....4-21

索引

このマニュアルについて

このマニュアルでは、BEA Tuxedo 環境で実行される CORBA アプリケーションをチューニングおよびスケーリングする方法について説明します。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 BEA Tuxedo CORBA アプリケーションのスケーリング」では、BEA Tuxedo 環境で実行される CORBA アプリケーションをスケーリングする方法について説明します。
- 「第 2 章 CORBA サーバ・アプリケーションのスケーリング」では、Production サンプル・アプリケーションを例として使用し、CORBA C++ サーバ・アプリケーションをスケーリングする方法について説明します。
- 「第 3 章 CORBA アプリケーションの分散」では、Production サンプル・アプリケーションおよび Bankapp サンプル・アプリケーションを例として使用し、アプリケーションを分散する方法について説明します。
- 「第 4 章 CORBA アプリケーションのチューニング」では、アプリケーションをチューニングして性能を最適化する方法について説明します。

対象読者

このマニュアルは、主に BEA Tuxedo CORBA 環境で実行されるスケーラブルな C++ アプリケーションをビルドするアプリケーション開発者を対象としています。BEA Tuxedo プラットフォームおよび C++ プログラミングに読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe 社の Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、および C++ プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで **docsupport-jp@bea.com** までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。

規則	項目
イタリック体	強調またはマニュアルのタイトルを示します。

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力するテキストも等幅テキストで表示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な語を示します。 例 : `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示します。 例 : `String *expr*`
大文字のテキスト	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
`{ }`	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。
`[ ]`	構文の行で、オプション項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f *file-list*]...[-l *file-list*]...`

規則	項目
|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。
. . .	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

1 BEA Tuxedo CORBA アプリケーションのスケーリング

ここでは、BEA Tuxedo CORBA アプリケーションのスケーリングの基本概念とそれを行うためのタスクの概要を説明します。次の内容について説明します。

- BEA Tuxedo CORBA アプリケーションのスケーリングについて
- オブジェクト状態管理の使用
- サーバ・プロセスおよびサーバ・グループの複製
- マルチスレッド・サーバの使用
- ファクトリ・ベース・ルーティングの使用 (CORBA サーバのみ)
- 受信するクライアント接続の多重化

BEA Tuxedo CORBA アプリケーションの詳細と例については、「第 2 章 CORBA サーバ・アプリケーションのスケーリング」を参照してください。

BEA Tuxedo CORBA アプリケーションのスケーリングについて

ここでは、次の内容について説明します。

- アプリケーションのスケーラビリティ要件
- BEA Tuxedo スケーラビリティの特徴
- オブジェクト状態管理の使用

アプリケーションのスケーラビリティ要件

多くのアプリケーションは、1 ～ 10 のサーバ・プロセスと 10 ～ 100 のクライアント・アプリケーションが動作している環境で適切に機能します。しかし、ビジネス環境のアプリケーションでは、数百の実行コンテキスト (コンテキストはこの場合はスレッドまたはプロセス)、数万のクライアント・アプリケーション、および数百万のオブジェクトを十分な性能水準でサポートしなければならない場合があります。

急激に増加する要求に晒されると、アプリケーションでは資源の不足や性能のボトルネックがすぐに明らかになります。したがって、スケーラビリティは BEA Tuxedo アプリケーションの極めて重要な特性です。

高度にスケーラブルな BEA Tuxedo アプリケーションは、次のようにしてビルドできます。

- 並列処理機能を追加して、BEA Tuxedo ドメインで複数のクライアント要求を同時に処理できるようにします。
- 複数のマシンでサーバ・アプリケーションの処理負荷を分担します。

BEA Tuxedo スケーラビリティの特徴

BEA Tuxedo では、以下の手段で大規模なアプリケーションのデプロイメントをサポートします。

- オブジェクト状態管理の最適化
- 複製されたサーバ・プロセスおよびサーバ・グループでのオブジェクトと要求のロード・バランシング
- マルチスレッド・サーバ (特定の種類のアプリケーションや処理環境で有効) の使用
- CORBA アプリケーションの場合は、ファクトリ・ベース・ルーティング
- データ依存型ルーティングの使用 (BEA Tuxedo ATMI のみ)
- 受信するクライアント接続の多重化

オブジェクト状態管理の使用

ここでは、次の内容について説明します。

- CORBA オブジェクト状態モデル
- 状態を持たないオブジェクトおよび状態を持つオブジェクトのインプリメント
- パラレル・オブジェクト

大規模なクライアント / サーバ・システムでは、スループットと応答時間を最適化することが重要であるため、オブジェクト状態管理は基本的な考慮事項となっています。オブジェクト状態管理の使用の詳細については、第 2 章の 4 ページ「状態を持たないオブジェクト・モデルの使用」および『CORBA 技術情報』の「Process-Entity デザイン・パターン」を参照してください。

CORBA オブジェクト状態モデル

BEA Tuxedo CORBA では、3 種類のオブジェクト状態管理モデルをサポートしています。

- メソッド・バウンド・オブジェクト
- プロセス・バウンド・オブジェクト
- トランザクション・バウンド・オブジェクト

これらのモデルの詳細については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』の「CORBA サーバ・アプリケーションの概念」を参照してください。

メソッド・バウンド・オブジェクト

メソッド・バウンド・オブジェクトは、クライアント呼び出しの有効期間中のみ、マシンのメモリにロードされています。呼び出しが完了すると、オブジェクトは非活性化され、そのオブジェクトの状態データはすべて、メモリからフラッシュされます。ここでは、メソッド・バウンド・オブジェクトは状態を持たないオブジェクトと見なされます。

メソッド・バウンド・オブジェクトを使用すると、アプリケーション内に状態を持たないサーバ・モデルを作成できます。状態を持たないサーバ・モデルを使用することで、活性化されたオブジェクトに送信済みの要求を、利用可能な任意のサーバへ移動できます。これにより、何千ものオブジェクト、さらには何百万ものオブジェクトの同時実行が可能になります。クライアント・アプリケーションのビューからは、すべてのオブジェクトがサービス要求に利用可能です。ただし、サーバ・アプリケーションがオブジェクトをメモリにマッピングするのが、クライアント呼び出しの有効期間中のみであるため、任意の時点でメモリ内にあるオブジェクトのうち、サーバ・アプリケーションによって管理されているものはほとんどありません。

プロセス・バウンド・オブジェクト

プロセス・バウンド・オブジェクトは、最初に呼び出されたときから、そのオブジェクトが実行されているプロセスがシャットダウンされるまで、メモリ内にとどまります。プロセス・バウンド・オブジェクトは、クライアント呼び出し時に活性化することも、クライアントが呼び出される前に、事前活性化オブジェクト

として明示的に活性化することもできます。アプリケーション側で、プロセス・バウンド・オブジェクトの非活性化を制御できます。ここでは、プロセス・バウンド・オブジェクトは状態を持つオブジェクトと見なされます。

しかるべき場合には、大量の状態データを備えたプロセス・バウンド・オブジェクトをメモリ内に残して、複数のクライアント呼び出しの処理を行うことができます。それにより、クライアント呼び出しのたびにオブジェクトの状態データを読み書きせずに済みます。

トランザクション・バウンド・オブジェクト

トランザクション・バウンド・オブジェクトも、状態を持つオブジェクトであると見なされます。なぜなら、トランザクションの範囲内においては、これらのオブジェクトは呼び出しと呼び出しの合間でもメモリ内に残存できるからです。オブジェクトがトランザクション範囲内で活性化された場合、そのオブジェクトはトランザクションがコミットまたはロールバックされるまでは、活性化されたままです。オブジェクトがトランザクション範囲外で活性化された場合、その振る舞いはメソッド・バウンド・オブジェクトと同様です。つまり、オブジェクトはクライアント呼び出しの有効期間中だけロードされています。

状態を持たないオブジェクトおよび状態を持つオブジェクトのインプリメント

一般に、アプリケーション開発者は状態を持たないオブジェクトのインプリメントと状態を持つオブジェクトのインプリメントのコストを均衡化する必要があります。

状態を持たないオブジェクトと状態を持つオブジェクトについて

状態を持たないオブジェクトを使うか、状態を持つオブジェクトを使うかの判断は、さまざまな要因に左右されます。永続状態を持つオブジェクトの初期化コストが高額の場合は、オブジェクトが会話中にアイドル状態であっても、状態を持ったままにしておくほうが合理的です。初期化コストが嵩む理由としてはたとえば、オブジェクトのデータが多くの領域を占めている、または永続状態が活性化を行うサーバントから非常に遠隔のディスクにあることなどが考えられます。

オブジェクトを活性化されたままにしておくコストが、マシンのリソース使用率との関連で高額になる場合は、そのようなオブジェクトは状態を持たないようにするほうが合理的です。

オブジェクトの状態を、アプリケーションに合わせて効率的かつ適切な方法で管理することにより、アプリケーションの能力を最大限に高めて、多数のオブジェクトを使用する多数のクライアント・アプリケーションを同時にサポートできます。オブジェクト状態の管理方法は、アプリケーション固有の特性や要件に応じて変わります。CORBA アプリケーションの場合、オブジェクト状態の管理は、これらのオブジェクトに `method` 活性化方針を割り当てることによって行います。この方針を割り当てると、アイドル状態のオブジェクトのインスタンスを非活性化して、マシン・リソースをほかのオブジェクト・インスタンスに割り当てられるようにする効果があります。

状態を持たないオブジェクトを使用すべき場合

サーバ・リソースはオブジェクトがアイドル状態のときには絶対に使用されないため、状態を持たないオブジェクトは一般に、サーバ・リソースの性能を高め、最適な使い方を実現します。状態を持たないオブジェクトの使用は、サーバ・アプリケーションのインプリメンテーションを行うのには良い手法です。特に、次の場合には適しています。

- クライアント・アプリケーションが、オブジェクトに対する呼び出しと呼び出しの合間にユーザ入力を待機する場合
- クライアント要求に、サーバ・アプリケーションが必要とするデータがすべて含まれており、サーバがそのデータのみを使用してクライアント要求を処理できる場合
- オブジェクトのアクセス率は高いが、ある特定の 1 つのクライアント・アプリケーションからのアクセス率は低い場合

状態を持たないオブジェクトにすることで、一般に、クライアント・アプリケーションからの入力待機中にサーバ・アプリケーションのリソースが無駄に予約されることは確実になくなります。

状態を持たないオブジェクト・モデルを採用したアプリケーションの特長は次のとおりです。

- 呼び出しとそれに関連する情報は、サーバ・アプリケーションがクライアント要求を実行し終わった後は保持されません。
- 受信されるクライアント要求は、最初に利用可能なサーバ・プロセスに送られます。要求が満たされると、アプリケーションの状態は消滅し、サーバ・

アプリケーションはほかのクライアント・アプリケーション要求に使用できるようになります。

- オブジェクトの永続状態の情報は、サーバ・プロセス外に存在します。このオブジェクトが呼び出されるたびに、永続状態がメモリに読み込まれます。
- 所定のクライアント・アプリケーションからの、オブジェクトに対する連続的な要求を、別のサーバ・プロセスによって処理できます。
- 通常、状態を持たないオブジェクトを実行しているマシンの全体的なシステム性能が向上します。

状態を持つオブジェクトを使用すべき場合

状態を持つオブジェクトは、いったん活性化されると、オブジェクトが存在しているプロセスがシャットダウンされたり、オブジェクトが活性化されているトランザクションが完了したりといった、特定のイベントが発生するまで、メモリ内にとどまります。

状態を持つオブジェクトの使用が推奨されるのは、以下のような場合です。

- オブジェクトが、寿命が長く、よく知られたオブジェクトのように、多数のクライアント・アプリケーションによって頻繁に使用されるものである場合。サーバ・アプリケーションがこれらのオブジェクトを活性化されたままにしていると、クライアント・アプリケーションがそれらにアクセスする際の応答時間は通常、最短化されます。これらの活性化されたオブジェクトは多くのクライアント・アプリケーションで共有されるため、このタイプのオブジェクトは比較的わずかしかメモリ内に存在しません。

  注記 オブジェクトがどのようにトランザクションに関与する可能性があるかは、慎重に考慮する必要があります。オブジェクトは、一時的または永続的に、特定プロセスにバインドできます。一時的にバインドされたものがトランザクション・バウンド・オブジェクト、永続的にバインドされたものがプロセス・バウンド・オブジェクトです。トランザクションに関与するオブジェクトを、別のクライアント・アプリケーションまたはオブジェクトから呼び出すことはできません。呼び出そうとした場合、BEA Tuxedo はおそらく、そのオブジェクトが使用中であることを示すエラーを返します。多数のクライアント・アプリケーションでの使用が意図されている、状態を持つオブジェクトが、頻繁にまたは長期にわたってトランザクションに関与している場合、それがボトルネックにつながる可能性があります。

- トランザクションを完了するために、クライアント・アプリケーションはオブジェクトのオペレーションを連続的に呼び出す必要があり、呼び出しと呼

び出しの合間でのユーザ入力待機中にアイドル状態になることはありません。呼び出しと呼び出しの間にオブジェクトが非活性化されると、各々の呼び出しの合間に状態が読み書きされるため、応答時間が長くなります。

状態を持つオブジェクトは、以下のような振る舞いをします。

- 状態情報はサーバ呼び出しの合間にも保持され、オブジェクトは通常、指定された期間中は、所定のクライアント・アプリケーションのために確保されたままです。データはクライアント・アプリケーションとサーバ・アプリケーションの間で送受信されますが、サーバ・プロセスにより追加のコンテキストまたはアプリケーション状態情報がメモリ内に保持されます。
- 1 つまたは複数の、状態を持つオブジェクトが大量のマシン・リソースを消費している場合、その状態を持つオブジェクトと関連していないタスクとプロセスについてのサーバの性能が、状態を持たないサーバ・モデルの場合よりも低くなる可能性があります。

  たとえば、オブジェクトがデータベースをロックしており、メモリ内に大量のデータをキャッシュしている場合、その状態を持つオブジェクトが使用している、そのデータベースとメモリは、トランザクション継続中、一貫してほかのオブジェクトに対しては使用不可となる可能性があります。

パラレル・オブジェクト

パラレル・オブジェクトは、定義上は状態を持たないオブジェクトなので、複数のサーバに同時に存在できます。BEA Tuxedo のリリース 8.0 では、インプリメンテーション・コンフィギュレーション・ファイル (ICF) を使用して、特定のインプリメンテーションのすべてのオブジェクトを強制的にパラレル・オブジェクトにすることができます。それにより、性能を向上させる効果が得られます。パラレル・オブジェクトの詳細については、第 1 章の 19 ページ「パラレル・オブジェクトの使用」を参照してください。

サーバ・プロセスおよびサーバ・グループの複製

ここでは、次の内容について説明します。

- サーバ・プロセスおよびサーバ・グループの複製について
- コンフィギュレーションのオプション
- サーバ・プロセスの複製
- サーバ・グループの複製

サーバ・プロセスとサーバ・グループの複製の詳細については、次のトピックを参照してください。

- 第 4 章の 5 ページ「複製されたサーバ・プロセスおよびグループのコンフィギュレーション」
- 第 2 章の 5 ページ「サーバ・プロセスおよびサーバ・グループの複製によるスケーリング」

サーバ・プロセスおよびサーバ・グループの複製について

BEA Tuxedo CORBA 環境では、CORBA オブジェクトを複数のサーバにわたるようにデプロイし、フェイルオーバーの信頼性を高めるとともに、クライアントの作業負荷をロード・バランシングによって分担できます。BEA Tuxedo CORBA ロード・バランシングは、デフォルトで有効になっています。ロード・バランシングのコンフィギュレーションの詳細については、第 4 章の 4 ページ「システム制御のロード・バランシングの有効化」を参照してください。BEA Tuxedo CORBA の機能を使用してのアプリケーション作業負荷の分散の詳細については、「第 3 章 CORBA アプリケーションの分散」を参照してください。

BEA Tuxedo アーキテクチャにより、次のようなサーバ編成が提供されます。

- グループ ー 個々のサーバを組み合わせると、グループを形成することができます。サーバのグループは、単一のマシン上で動作します。通常、グループ内のサーバは共通のリソース (データベースなど) にアクセスします。
- ドメイン ー マシンを組み合わせると、ドメインを形成できます。ドメインは、中央管理されます。複数のドメインは、個別に管理されます。ドメインどうしを相互に接続して、1 つのドメインから別のドメインへ、要求を透過的にルーティングすることもできます。ただし、各ドメインは、単独管理されます。

このアーキテクチャにより、アプリケーションを需要の多い期間または少ない期間に適合させたり、アプリケーションに対して要求される内部変更に対処したりするために、新規のサーバ、グループ、またはマシンを動的に追加または削除できます。BEA Tuxedo の実行時に、使用可能な各サーバ間で要求をルーティングすることにより、ロード・バランシングとフェイルオーバーがもたらされます。

システム管理者は、次の処理を行うことによって、BEA Tuxedo アプリケーションをスケーリングできます。

- サーバ・プロセスの複製。グループ内でより多くの活性化されたオブジェクト、およびサーバ間でのロード・バランシングをサポートするよう、サーバ・プロセスの数を増やします。
- サーバ・グループの複製。BEA Tuxedo が、複数のサーバ・マシン間に要求の処理を分散することによってロード・バランシングを行えるよう、サーバ・グループの数を増やします。

コンフィギュレーションのオプション

サーバ・アプリケーションは、以下のようにコンフィギュレーションできます。

- 1 つまたは複数のサーバ・プロセスが、1 つまたは複数のインターフェイスをインプリメントする、単一のマシン。サーバは、シングル・スレッドでもマルチスレッドでもかまいません。
- 複数のサーバ・プロセスと複数のインターフェイスを備えた複数のマシン。

サーバ・プロセスを複製するか、スレッドを追加することにより、クライアント / サーバ・アプリケーションの並列処理機能を強化できます。サーバ・グループを追加して、リソース・マネージャ間で処理を分担することができます。

CORBA アプリケーションでは、第 1 章の 16 ページ「ファクトリ・ベース・ルーティングの使用 (CORBA サーバのみ)」で説明したファクトリ・ベース・ルーティングをインプリメントできます。

サーバ・プロセスの複製

システム管理者は、サーバ・ノード上でより多くの活性化されたオブジェクトを同時にサポートしたり、より多くの要求を同時に処理したりするために、サーバを複製することによって、アプリケーションをスケーリングできます。複製サーバ・プロセスのコンフィギュレーションについては、第 4 章の 5 ページ「複製されたサーバ・プロセスおよびグループのコンフィギュレーション」を参照してください。

注記 BEA Tuxedo のリリース 8.0 では、活性化されたオブジェクトについて、ユーザが制御する同時実行モデルをサポートしています。同時実行方針機能の説明については、第 1 章の 8 ページ「パラレル・オブジェクト」を参照してください。

利点

複製サーバ・プロセスを使用する利点には、以下のようなものがあります。

- 受信する要求のロード・バランシング。
- グループ内の任意のサーバに対するクライアント要求の処理。サーバ・グループの BEA Tuxedo ドメインに要求が到達すると、BEA Tuxedo はその要求をそのグループ内で最も負荷の軽いサーバ・プロセスにルーティングします。
- 複数のサーバ・プロセスを使用したサーバ・アプリケーション性能の向上。1 つのサーバ・プロセスが一度に 1 つのクライアント要求を処理するのではなく、複数のサーバ・プロセスで複数のクライアント要求を同時に処理できます。
- サーバ・プロセスの 1 つが停止した場合のフェイルオーバー保護の提供。

ガイドライン

複製サーバ・プロセスを使用する利点を最大限に高めるには、サーバ・アプリケーションによってインスタンス化された CORBA オブジェクトに、必ず一意のオブジェクト ID を付けます。これにより、オブジェクトに対するクライアント呼び出しで、オブジェクトを活性化済みオブジェクトのキューに入れるのではなく、必要に応じて利用可能なサーバ・プロセス数の制限内においてインスタンス化できます。

また、アプリケーション・パターンおよび処理環境の種類によっては、複数のプロセスを使用してアプリケーションの回復機能を改善するか、スレッドを使用して性能を高めるかの、兼ね合いを考える必要があります。

フェイルオーバーが改善されるのは、スレッドではなくプロセスを追加した場合のみです。シングル・スレッド・サーバおよびマルチスレッド・サーバの使用については、第 1 章の 14 ページ「マルチスレッド CORBA サーバを使用すべき場合」を参照してください。

サーバ・グループの複製

サーバ・グループは、BEA Tuxedo に固有のもので、BEA Tuxedo のスケーラビリティ機能の基幹部分です。グループは、単一ノード上の 1 つまたは複数のサーバで構成されます。システム管理者は、サーバ・グループを複製し、ドメイン内のロード・バランシングをコンフィギュレーションすることにより、BEA Tuxedo アプリケーションをスケーリングできます。

サーバ・グループを複製するには、同じタイプのサーバとリソース・マネージャを備えた別のサーバ・グループを定義して、共有リソース (データベースなど) への並行アクセスを実現することが必要です。たとえば CORBA アプリケーションでは、ファクトリ・ベース・ルーティングを使用して、データベースのパーティション間で処理を分担できます。

`UBBCONFIG` ファイルにより、どのようにサーバ・グループをコンフィグレーションするか、およびそれらのグループが稼動する場所が指定されます。複数のサーバ・グループを使用すると、BEA Tuxedo は次のことを行えます。

- 所定のアプリケーションまたはアプリケーションのセットの処理負荷を、追加のマシン間で分担する。

- CORBA アプリケーションで、ファクトリ・ベース・ルーティングを使用して所定のインターフェイス上の要求の 1 セットを 1 つのグループに送信し、同じインターフェイス上の要求の別のセットを別のグループに送信する。

複製サーバ・グループのコンフィギュレーションについては、第 4 章の 5 ページ「複製されたサーバ・プロセスおよびグループのコンフィギュレーション」を参照してください。

マルチスレッド・サーバの使用

ここでは、次の内容について説明します。

- マルチスレッド CORBA サーバについて
- マルチスレッド CORBA サーバを使用すべき場合
- コーディングの推奨事項
- マルチスレッド CORBA サーバのコンフィギュレーション

マルチスレッド処理を行うためのサーバのコンフィギュレーション方法については、第 4 章の 6 ページ「マルチスレッド・サーバのコンフィギュレーション」を参照してください。

マルチスレッド CORBA サーバについて

システム管理者は、CORBA サーバ内でマルチスレッド処理を有効にし、アプリケーションの UBBCONFIG ファイルのコンフィギュレーション・パラメータ (作成可能なサーバ・スレッドの最大数) をチューニングすることによって、BEA Tuxedo アプリケーションのスケーリングを行えます。

BEA Tuxedo CORBA では、マルチスレッド CORBA アプリケーションのコンフィギュレーション機能をサポートしています。シングル・スレッド CORBA サーバでは一度に 1 つの要求しか実行できませんが、マルチスレッド CORBA サーバでは複数のオブジェクト要求の同時処理を行えます。

サーバ・スレッドは、アプリケーション・プログラムではなく、BEA Tuxedo CORBA ソフトウェアによって開始および管理されます。内部では、BEA Tuxedo CORBA が、利用可能なサーバ・スレッドのプールを管理しています。CORBA サーバがマルチスレッド処理用にコンフィグレーションされている場合は、クライアント要求が受信されると、スレッド・プールからの利用可能なサーバ・スレッドがその要求を実行するように、スケジューリングが行われます。オブジェクトが活性化されている間、スレッドは使用中です。要求が完了すると、スレッドは利用可能なスレッドのプールに返されます。

マルチスレッド CORBA サーバを使用すべき場合

複数の独立したスレッドを使用するようにアプリケーションを設計すると、アプリケーション内に並行性がもたらされ、総合的なスループットを改善できます。複数のスレッドを使用すると、各スレッドが複数の独立したタスクを並列に処理する効率的なアプリケーションを構築できます。マルチスレッドは、以下の場合に特に役立ちます。

- ほかの処理に必ずしも依存しない一連の長いオペレーションが存在する。
- 共有されるデータの量が少なく、識別可能である。
- 並列に実行できる複数のアクティビティにタスクを分割できる。
- オブジェクトがリエントラントでなければならないときが存在する。

コンピュータ・オペレーションの中には、完了するのに長い時間を要するものがあります。マルチスレッド・アプリケーションの設計では、要求とオペレーション完了の間の待ち時間を、大幅に短縮できます。これは、オペレーションが数多くの I/O オペレーションを実行する環境に当てはまります。たとえば、データベースにアクセスするとき、リモート・オブジェクトのオペレーションを呼び出すとき、マルチ・プロセッサ・マシン上の CPU バウンドなどです。サーバ・プロセスでマルチスレッドをインプリメントすると、サーバが一定時間に処理できる要求の数が増加します。

マルチスレッド・サーバ・アプリケーションの主要な要件は、複数のクライアント要求を同時に処理することです。マルチスレッド・サーバの使用の要件と利点の詳細については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

コーディングの推奨事項

クライアント・アプリケーションまたはサーバ・アプリケーションが、メッセージをユーザ・ログ (ULOG) に送る場合、マルチスレッド・サーバの性能を分析できるようにするには、各メッセージに次の識別子のうち 1 つを含めます。

- オブジェクト ID
- トランザクション ID (オブジェクトがトランザクションに関与している場合)

マルチスレッド CORBA サーバのコンフィギュレーション

マルチスレッド CORBA サーバをコンフィギュレーションするには、アプリケーションの UBBCONFIG ファイルの設定を変更します。マルチスレッド・サーバをインプリメントするための UBBCONFIG パラメータの定義については、第 4 章の 6 ページ「マルチスレッド・サーバのコンフィギュレーション」を参照してください。

ファクトリ・ベース・ルーティングの使用 (CORBA サーバのみ)

ここでは、次の内容について説明します。

- ファクトリ・ベース・ルーティングについて
- ファクトリ・ベースのインプリメンテーションのしくみ
- UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション

このトピックでは、BEA Tuxedo CORBA アプリケーションにおけるファクトリ・ベース・ルーティングについて概説します。ファクトリ・ベース・ルーティングの使用の詳細については、第 2 章の 12 ページ「UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション」を参照してください。

ファクトリ・ベース・ルーティングについて

ファクトリ・ベース・ルーティングを使用すると、オブジェクト・リファレンスと関連付けられたサーバ・グループを指定できます。その結果、所定のオブジェクトがインスタンス化されているグループとマシンを定義し、その後、所定のアプリケーションの処理負荷を、複数のマシン間に分散できます。

ファクトリ・ベース・ルーティングでは、ファクトリがオブジェクト・リファレンスを作成したときにルーティングが行われます。ファクトリは、オブジェクト・リファレンスを作成するための BEA Tuxedo CORBA TP フレームワークへの呼び出しにおいて、フィールド情報を指定します。TP フレームワークは、アプリケーションの `UBBCONFIG` ファイルの `ROUTING` セクションで定義されたルーティング基準に基づき、ルーティング・アルゴリズムを実行します。その結果得られるオブジェクト・リファレンスは、オブジェクト・リファレンスに対するメソッド呼び出し処理に適したサーバ・グループをターゲットとしています。このサーバ・グループにおけるインターフェイスをインプリメントするサーバはすべて、オブジェクト・リファレンスのためのサーバントを活性化するものとして適しています。

したがって、CORBA オブジェクトの活性化は、定義された基準に基づいてサーバ・グループにより分散可能であり、CORBA インターフェイスのさまざまなインプリメンテーションを、さまざまなグループに提供することができます。つまり、定義済みのグループ固有の差異に基づき、複数のサーバ・グループにわたって、同一の CORBA インターフェイスを複製することができます。

ファクトリ・ベース・ルーティングの主な利点は、アプリケーションのスケーリングを行うための単純な手段と、拡大しつつあるデプロイメント環境全体にわたる、特定のインターフェイスに対する呼び出しが得られることです。アプリケーションのデプロイメントを、追加のマシン間に分散することは、あくまでも管理上の機能であり、アプリケーションのコーディングやビルドをやり直す必要はありません。

ファクトリ・ベース・ルーティングの特性

ファクトリ・ベース・ルーティングには、次の特性があります。

- ファクトリ・オブジェクトのインプリメンテーションは、アプリケーション固有のルーティング情報を指定することにより、作成された CORBA オブジェクトの配置を間接的に制御できます。
- 第 2 章の 12 ページ「UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション」に示すように、特定の CORBA インターフェイスのインプリメンテーションが、複数のサーバ・プロセスに存在できます。
- 複数の CORBA インターフェイスが、単一のサーバ・グループ内に存在できます。
- 特定のサーバ・グループ内のサーバ・プロセスが、すべて同じ CORBA インターフェイスを使用する必要はありません。
- グループ内の所定のインターフェイスを提供するオブジェクト・インスタンスはすべて、同じバージョンのインプリメンテーションをサポートしている必要があります。
- ルーティングは、掲示板の基準を使用し、サーバ呼び出し中に発生します。

ファクトリ・ベースのインプリメンテーションのしくみ

ファクトリ・ベース・ルーティングをインプリメントするには、ファクトリによるオブジェクト・リファレンスの作成方法を変更する必要があります。まず、システム設計者との間で調整を行い、ルーティングの基準として使用するフィールドと値を決定します。次に、各インターフェイスについて、グループ ID の決定に使用されるルーティング基準を表すパラメータが、ファクトリのインターフェイス定義で指定されるように、ファクトリ・ベース・ルーティングをコンフィギュレーションする必要があります。

ファクトリ・ベース・ルーティングをコンフィギュレーションするには、UBBCONFIG ファイルで以下の情報を定義します。

- INTERFACES セクションの CORBA インターフェイスのルーティング基準識別子
- GROUPS セクションの、システムの分散に必要とされる数のサーバ・グループ

■ROUTING セクションのルーティング基準

- 必要に応じて、グループ、マシンおよびデータベース

注記 ファクトリ・ベース・ルーティングをインプリメントする際には、所定のインターフェイスと OID を備えたオブジェクトは、2 つの異なるグループの双方に同じオブジェクトのインプリメンテーションが含まれている場合、2 つのグループ内で同時に活性化されることに留意してください。これは、ファクトリが一意の OID を生成している場合は回避できます。所定のインターフェイス名および OID のオブジェクト・インスタンスが、ドメイン内では必ず一度に 1 つしか使用されないようにするには、次の処理のいずれかを行います。

- ファクトリ・ベース・ルーティングを使用して、特定の OID を持つオブジェクトは常に同じグループにルーティングされるようにする。
- 所定のオブジェクトのインプリメンテーションが、1 つのグループ内にしか存在しないように、ドメインをコンフィギュレーションする。

複数のクライアントが、所定のインターフェイス名と OID を含むオブジェクト・リファレンスを持っている場合、そのリファレンスは常に同じオブジェクト・インスタンスにルーティングされます。

その後のオブジェクト・リファレンスには、ターゲット・サーバが存在する場所を示すための追加情報が含まれます。ファクトリ・ベース・ルーティングは、各 CORBA オブジェクトについて 1 回ずつ、オブジェクト・リファレンスの作成時に行われます。

UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション

ルーティング基準により、特定のサーバ・グループに要求をルーティングするためのデータ値が指定されます。ファクトリ・ベース・ルーティングをコンフィギュレーションするには、UBBCONFIG ファイルの ROUTING セクションで、要求がルーティングされる各インターフェイスについて、ルーティング基準を定義します。ファクトリ・ベース・ルーティングのコンフィギュレーションの詳細については、第 2 章の 12 ページ「UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション」を参照してください。

複数ドメインにわたってファクトリ・ベース・ルーティングをコンフィギュレーションするには、現在の (ローカル) ドメインで使用されているが、別の (リモート) ドメインに存在するオブジェクトを識別するために、factory_finder.ini ファイルのコンフィギュレーションも行う必要があります。詳細については、『BEA Tuxedo Domains コンポーネント』の「複数の CORBA ドメインを設定する」を参照してください。

パラレル・オブジェクトの使用

ここでは、次の内容について説明します。

- パラレル・オブジェクトについて
- パラレル・オブジェクトのコンフィギュレーション

パラレル・オブジェクトについて

パラレル・オブジェクトのサポートは、BEA Tuxedo のリリース 8.0 で性能強化の一環として追加されています。パラレル・オブジェクト機能を利用すると、特定アプリケーションのすべてのビジネス・オブジェクトを状態を持たないオブジェクトとして指定できます。1 つのドメインの 1 つのサーバでしか実行できない、状態を持つビジネス・オブジェクトと異なり、状態を持たないビジネス・オブジェクトは 1 つのドメインのすべてのサーバで実行できます。パラレル・オブジェクトの利点は以下のとおりです。

注記 パラレル・オブジェクト機能を有効にするには、ICF ファイルで同時実行方針のオプションを `user_controlled` に設定します。詳細については、第 1 章の 23 ページ「パラレル・オブジェクトのコンフィギュレーション」を参照してください。

- 状態を持たないパラレル・オブジェクトは、同じドメインの複数のサーバで同時に実行できます。その結果、すべてのサーバを利用して同時に複数の要求を処理できるので性能が向上します。
- BEA Tuxedo は、パラレル・ビジネス・オブジェクトへの要求を処理するときには常に、ローカル・マシンで利用可能なサーバを最初に探します。ローカル・マシン上のすべてのサーバが、要求されたビジネス・オブジェクトの処理で使用中の場合、BEA Tuxedo はローカル・ドメイン内のほかのマシンで利用可能なサーバを探します。したがって、ローカル・マシンに複数のサーバがある場合は、ネットワーク・トラフィックが削減され、性能が向上します。

図 1-1 で示されるように、状態を持つビジネス・オブジェクトがマシン 2 のサーバ上で活性化されている場合、その後、そのビジネス・オブジェクトへの要求はすべてマシン 2 のグループ 2 に送信されます。マシン 2 上の活性化されたオブジェクトが、別の要求の処理に使用中であった場合、その要求はキューに入れられます。ビジネス・オブジェクトがマシン 2 上の要求の処理を停止した後でも、その状態を持つビジネス・オブジェクトに対するその後の要求はやはり、すべてグループ 2 に送信されます。オブジェクトがマシン 2 上で非活性化されると、その後の要求はマシン 2 上のグループ 2 に送られ、グループ 2 のほかのサーバによって処理可能です。

図 1-1　状態を持つビジネス・オブジェクトの使用

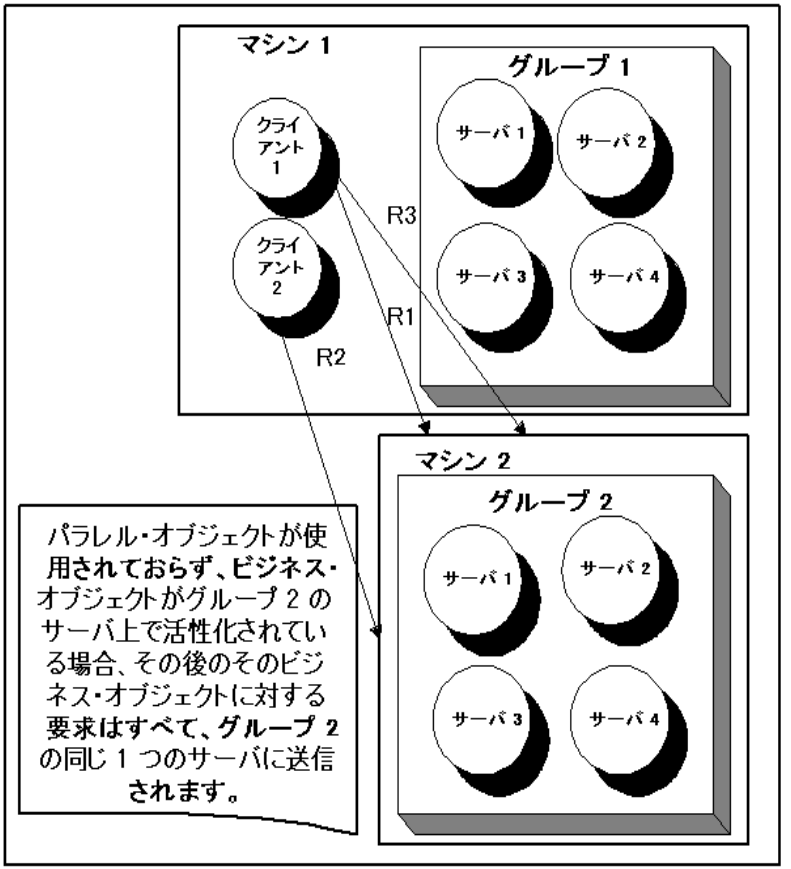
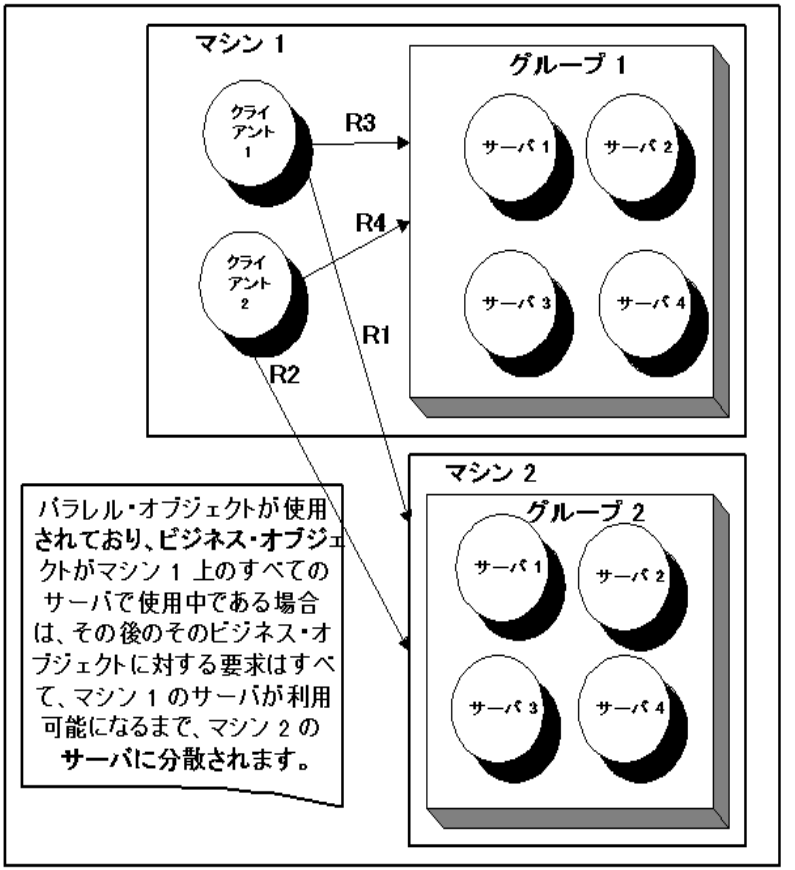


図 1-2 で示されるように、パラレル・オブジェクトがマシン 1 上のグループ 1 に含まれるすべてのサーバ上で実行されている場合、そのビジネス・オブジェクトに対するその後の要求はマシン 2 に送信され、グループ 1 でサーバが使用可能になるまで、グループ 2 のサーバに分散されます。状態を持たない、ユーザに制御されるビジネス・オブジェクトの複数のインスタンスは、複数のサーバ上で同時に実行可能です。ローカル・マシンで利用可能なサーバがある限りは、要求はマシン 1 上のサーバに分散されます。ただし、BEA Tuxedo のロード・バランシング機能が、サーバに対する負荷を理由に、要求をグループ 2 のサーバで処理すべきであると判断した場合には、その限りではありません。この判断を行うために、ロード・バランシング機能では LOAD パラメータを使用します。これは、UBBCONFIG ファイルの INTERFACES セクションで設定されます。

図 1-2　状態を持たないビジネス・オブジェクトの使用



LOAD パラメータについては、第 3 章の 12 ページ「INTERFACES セクションの変更」を参照してください。

パラレル・オブジェクトのコンフィギュレーション

パラレル・オブジェクトのサポートは、BEA Tuxedo のリリース 8.0 から追加されました。特定の CORBA アプリケーションのパラレル・オブジェクトをインプリメントするには、ICF ファイルを使用します。ICF には、その ICF ファイル

が適用されるアプリケーションでインプリメントされているすべてのビジネス・オブジェクトを状態を持たないオブジェクトに設定する、ユーザ制御の同時実行方針のオプションが含まれます。

同時実行方針は、あるオブジェクトが一度に 1 つのサーバでのみ活性化されることを保証するために、アクティブ・オブジェクト・マップ (AOM) を使用するかどうかを決定します。旧リリースでは、AOM の使用はオプションではなく必須でした。AOM の使用は、システム制御の同時実行と呼ばれます。システム制御の同時実行モデルと異なり、AOM を使用しないユーザ制御のモデルでは、一度に複数のサーバで、同じオブジェクトを活性化することができます。したがって、ユーザ制御の同時実行を使用すると、性能とロード・バランシングを改善できます。パラレル・オブジェクトのためのユーザ制御の同時実行のコンフィギュレーションの詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「パラレル・オブジェクト」を参照してください。

受信するクライアント接続の多重化

ここでは、次の内容について説明します。

- IIOP リスナ / ハンドラ
- ISH プロセス数の増加

システム管理者は、UBBCONFIG ファイルで、アプリケーション・サイトによってサポートされている受信するクライアント接続数を増やすことによって、BEA Tuxedo アプリケーションをスケーリングできます。BEA Tuxedo は、リスナ / ハンドラのマルチコンテキスト化された、複数の状態を持つゲートウェイを提供して、クライアントによって発行されたすべての要求の多重化処理を行います。

IIOP リスナ / ハンドラ

IIOP リスナ (ISL) を使うと、IIOP を使用するリモート BEA Tuxedo CORBA クライアントによる BEA Tuxedo CORBA オブジェクトへのアクセスが可能になります。ISL とは、IIOP 接続を要求する リモート CORBA クライアントをリッスンするプロセスです。IIOP ハンドラ (ISH) は、リモート CORBA クライアントの

代理プロセスとして作用するマルチプレクサ・プロセスです。ISL と ISH はどちらも、アプリケーション・サイト上で実行されます。アプリケーション・サイトは、1 つまたは複数の ISL プロセスと、複数の関連する ISH プロセスを持つことができます。各 ISH は、単一の ISL と関連付けられます。

クライアントは、既知のネットワーク・アドレスを使用して ISL プロセスに接続します。ISL は、利用可能な中から最適な ISH を選択して、接続を直接それに渡すことにより、ISH プロセス間でロード・バランシングを行います。ISL/ISH は、アプリケーション・クライアントの代わりに、コンテキストを管理します。ISL と ISH の詳細については、『BEA Tuxedo のファイル形式とデータ記述方法』の ISL の説明を参照してください。

ISH プロセス数の増加

システム管理者は、アプリケーション・サイト上の ISH プロセスの数を増加することによって、BEA Tuxedo CORBA アプリケーションをスケーリングできます。それにより、ISL でのロード・バランシングを、より多くの ISH プロセス間で行えるようになります。デフォルトでは、ISH は最高で 10 個のクライアント接続を処理できます。この数を増やすには、オプションの `CLOPT -x` *`mpx-factor`* パラメータを ISL コマンドに渡します。その際、*`mpx-factor`* で、各 ISH が処理できる ISH クライアント接続数 (最大 4096)、つまり ISH の多重化のレベルを指定します。ISH プロセス数を増やすと、アプリケーション・サイトで同時に処理するプロセスの数が増えるので、アプリケーションの性能に影響が出る可能性があります。

システム管理者は、そのほかの ISH オプションもチューニングして、BEA Tuxedo アプリケーションをスケーリングできます。詳細については、『BEA Tuxedo のファイル形式とデータ記述方法』の ISL の説明を参照してください。

2 CORBA サーバ・アプリケーションのスケーリング

ここでは、以下の内容について説明します。

- Production サンプル・アプリケーションのスケーリングについて
- OMG IDL の変更
- 状態を持たないオブジェクト・モデルの使用
- サーバ・プロセスおよびサーバ・グループの複製によるスケーリング
- ファクトリ・ベース・ルーティングによるスケーリング
- 設計上の追加考慮事項
- アプリケーションをさらにスケーリングする方法

このトピックでは Production サンプル・アプリケーションを例として使用し、CORBA C++ アプリケーションをスケーリングして処理能力を高める方法を示します。事前に、必ず次の個所をお読みください。

- BEA Tuxedo CORBA アプリケーションのチューニングとスケーリングに関する包括的な説明については、「第 1 章 BEA Tuxedo CORBA アプリケーションのスケーリング」
- BEA Tuxedo オンライン・マニュアルの「Production サンプル・アプリケーション」

Production サンプル・アプリケーションのスケーリングについて

Production サンプル・アプリケーションは、Wrapper サンプル・アプリケーションと同じエンド・ユーザ機能を提供します。Production サンプル・アプリケーションでは、BEA Tuxedo ソフトウェアの機能を使用して既存の BEA Tuxedo アプリケーションをスケーリングする方法が示されます。

この節では、以下の内容について説明します。

- 設計上の目標
- アプリケーションのスケーリング方法

設計上の目標

Production サンプル・アプリケーションの設計上の主な目標は、次の処理により、対応できるクライアント・アプリケーションの数を大幅に増やすことです。

- 並列的に、および 1 つのマシン上で、同じインターフェイスをインプリメントする複数のオブジェクトに対するクライアント要求を処理します。
- 1 つのマシンへある特定の学生ために、および別のマシンへ別の学生のために、要求を転送します。
- マシンをさらに追加して、処理負荷を分担します。

アプリケーションのスケーリング方法

これらの設計上の目標に対応するため、Production サンプル・アプリケーションは次のようにしてスケーリングされています。

- 状態を持たないオブジェクト・モデルをインプリメントして、サーバ・プロセスが同時に管理できるクライアント要求の数を増やします。
- University、Billing および BEA Tuxedo Teller アプリケーションのサーバ・プロセスを、それらのコンフィギュレーションが行われているグループ内で

複製します。該当するグループは、UBBCONFIG ファイルで定義されている ORA_GRP サーバ・グループおよび APP_GRP サーバ・グループです。

- ORA_GRP サーバ・グループおよび APP_GRP サーバ・グループを追加のサーバ・マシンである Production マシン 2 上で複製し、データベースの分割も行います。
- 以下のオブジェクトに一意のオブジェクト ID (OID) を割り当て、それらがそれぞれのグループにおいて同時に複数回インスタンス化されるようにします。
  - RegistrarFactory
  - Registrar
  - TellerFactory
  - Teller

  これにより、これらのオブジェクトはプロセスではなくクライアント・アプリケーションごとに利用可能となるので、並列処理機能に対応できるようになります。
- ファクトリ・ベース・ルーティングをインプリメントして、1 つのマシンへは一部の学生のために、別のマシンへはほかの学生のために、クライアント要求を転送します。

注記 Production サンプル・アプリケーションの使用を簡単にするには、1 つのデータベースを使い、1 つのマシン上で実行されるように、アプリケーションを BEA Tuxedo ソフトウェア・キット上でコンフィギュレーションします。ただし、この章で示した例では、アプリケーションは 2 つのデータベースを使って 2 つのマシン上で動作しています。

Production サンプル・アプリケーションは、数台のマシン上で動作し、複数のデータベースを使用すべくコンフィギュレーションできるよう設計されています。コンフィギュレーションを複数のマシンとデータベース用に変更するには、UBBCONFIG ファイルを修正して、データベースを分割することが必要です。この手順は、第 2 章の 23 ページ「アプリケーションをさらにスケーリングする方法」で説明しています。

以下の節では、Production サンプル・アプリケーションが複製されたサーバ・プロセスとサーバ・グループ、オブジェクト状態管理、およびファクトリ・ベース・ルーティングをどのように使用して、スケーラビリティの目標を達成するかを説明します。

OMG IDL の変更

Production サンプル・アプリケーションでの OMG IDL の変更は、`RegistrarFactory` オブジェクトの `find_registrar()` オペレーション、および `TellerFactory` オブジェクトの `find_teller()` オペレーションに限定されています。2 つのオペレーションは、それぞれ学生 ID と口座番号を要求するように修正する必要があります。これらは、ファクトリ・ベース・ルーティングのインプリメントに必要なものです。Production サンプル・アプリケーションでのファクトリ・ベース・ルーティングのインプリメンテーションと使用については、第 2 章の 11 ページ「ファクトリ・ベース・ルーティングによるスケーリング」を参照してください。

状態を持たないオブジェクト・モデルの使用

ここでは、Production サンプル・アプリケーションにおいて、スケーラビリティを増大させるために、オブジェクト状態管理をどのように `Registrar` オブジェクトおよび `Teller` オブジェクトで使用するかを説明します。オブジェクト状態管理の概要については、第 1 章の 3 ページ「オブジェクト状態管理の使用」を参照してください。

スケーラビリティを増大させるには、Production サーバ・アプリケーションで、`Registrar` および `Teller` オブジェクトに `method` 活性化方針をコンフィギュレーションします。これら 2 つのオブジェクトに `method` 活性化方針を割り当てた結果、振る舞いに次の変化がみられます。

- これらのオブジェクトが呼び出されるときは常に、適切なサーバ・グループ内の BEA Tuxedo ドメインによってインスタンス化されます。
- 呼び出しが完了すると、BEA Tuxedo ドメインによりこれらのオブジェクトは非活性化されます。

Basic から Wrapper までの各種サンプル・アプリケーションでは、`Registrar` オブジェクトはプロセス・バウンド・オブジェクト (`process` 活性化方針) となっていました。`Registrar` オブジェクトに対するクライアント要求はすべて、常

にサーバ・マシンのメモリ内の同じオブジェクト・インスタンスに送られました。Basic サンプル・アプリケーションの設計は、すべての小規模なデプロイメントに適しています。しかし、クライアント・アプリケーションの要求が増すにつれて、Registrar オブジェクト上のクライアント要求はキューに入れられるようになり、このため応答時間が遅くなります。

ただし、Registrar および Teller オブジェクトが状態を持たないオブジェクト (method 活性化方針) であり、これらのオブジェクトを管理するサーバ・プロセスが複製されている場合、Registrar および Teller の各オブジェクトは並行して複数のクライアント要求を処理できます。これらのオブジェクトで処理できる同時のクライアント要求の数で制約があるのは、Registrar および Teller のオブジェクトをインスタンス化できる、利用可能なサーバ・プロセスの数だけです。したがって、これらの状態を持たないオブジェクトはマシン・リソースのより効率的な使用と、クライアント応答時間の短縮を実現します。

最も重要なのは、BEA Tuxedo CORBA で、Registrar および Teller オブジェクトのコピーを各複製サーバ・プロセス内でインスタンス化できるように、オブジェクトの各コピーが一意でなければならないことです。これらのオブジェクトの各インスタンスを一意にするために、オブジェクトのファクトリでは一意のオブジェクト ID をオブジェクトに割り当てる必要があります。

BEA Tuxedo アプリケーションが複製サーバ・アプリケーション・プロセスのそれぞれで Registrar および Teller オブジェクトのコピーをインスタンス化するには、オブジェクトの各コピーが一意のオブジェクト ID (OID) を備えていなければなりません。これらのオブジェクトを作成するファクトリが、一意の OID を割り当てる役割を担います。一意のオブジェクト ID の生成については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。その他の設計上の考慮事項については、第 2 章の 18 ページ「設計上の追加考慮事項」を参照してください。

サーバ・プロセスおよびサーバ・グループの複製によるスケーリング

ここでは、次の内容について説明します。

- Production アプリケーションでのサーバ・プロセスの複製
- Production アプリケーションでのサーバ・グループの複製

- Production アプリケーションの複製サーバ・プロセスおよびグループのコンフィギュレーション

このトピックでは、サーバ・プロセスおよびサーバ・グループを複製することによる、Production サンプル・アプリケーションのスケーリング方法を説明します。概要については、第 1 章の 9 ページ「サーバ・プロセスおよびサーバ・グループの複製」を参照してください。

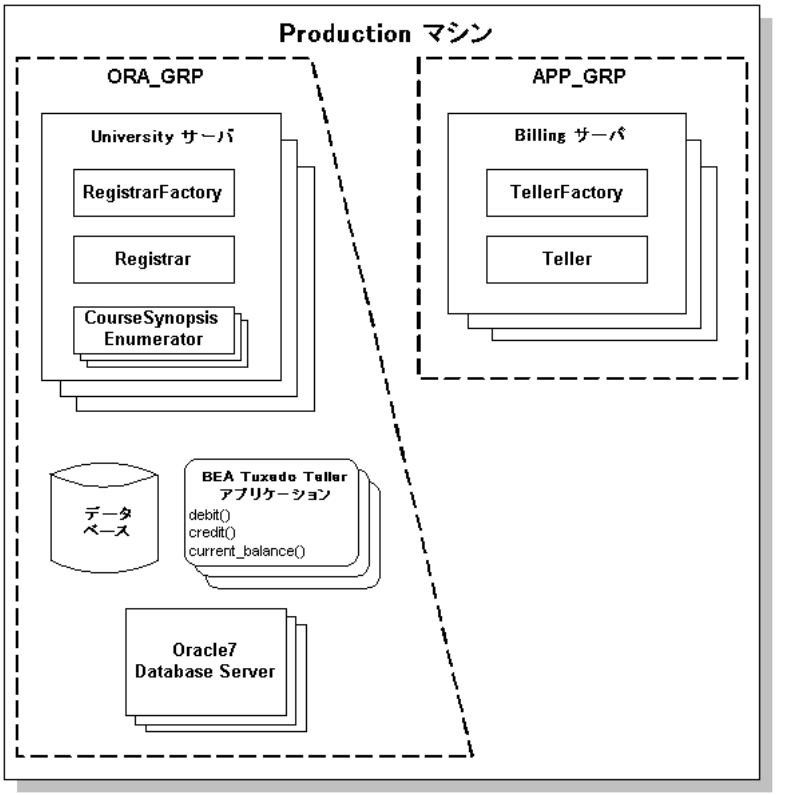
Production アプリケーションでのサーバ・プロセスの複製

この節では、Production サンプル・アプリケーションによるサーバ・アプリケーションの複製方法を説明します。この機能の概要については、第 1 章の 11 ページ「サーバ・プロセスの複製」を参照してください。

図 2-1 では、単一のマシン上で実行される、複製された ORA_GRP および APP_GRP グループを示します。

- University サーバ・アプリケーション、BEA Tuxedo Teller アプリケーション、および Oracle7 TMS サーバ・プロセスは、ORA_GRP グループ内で複製されます。
- Billing サーバ・プロセスは、APP_GRP グループ内で複製されます。

図 2-1 Production サンプルの複製サーバ・グループ



これらのグループの 1 つに要求が到達したとき、BEA Tuxedo ドメインにはこの要求を処理できるサーバ・プロセスがいくつかあります。BEA Tuxedo ドメインは、最も負荷の軽いサーバ・プロセスを選択することができます。

図 2-1 では、次の点に留意してください。

- 常に、特定のサーバ・プロセス内で、`RegistrarFactory`、`Registrar`、`TellerFactory`、または `Teller` オブジェクトのインスタンスが複数存在することはできません。
- `CourseSynopsisEnumerator` オブジェクトは、どの University サーバ・プロセスでも、任意の数だけ存在することができます。

Production アプリケーションでのサーバ・グループの複製

この節では、Production サンプル・アプリケーションによるサーバ・グループの複製方法を説明します。この機能の概要については、第 1 章の 12 ページ「サーバ・グループの複製」を参照してください。

図 2-2 は、アプリケーションの UBBCONFIG ファイルで ORA_GRP2 および APP_GRP2 として指定された、別のマシン上で複製される Production サンプル・アプリケーション・グループを示します。

図 2-2　複数のマシンにわたるサーバ・グループの複製

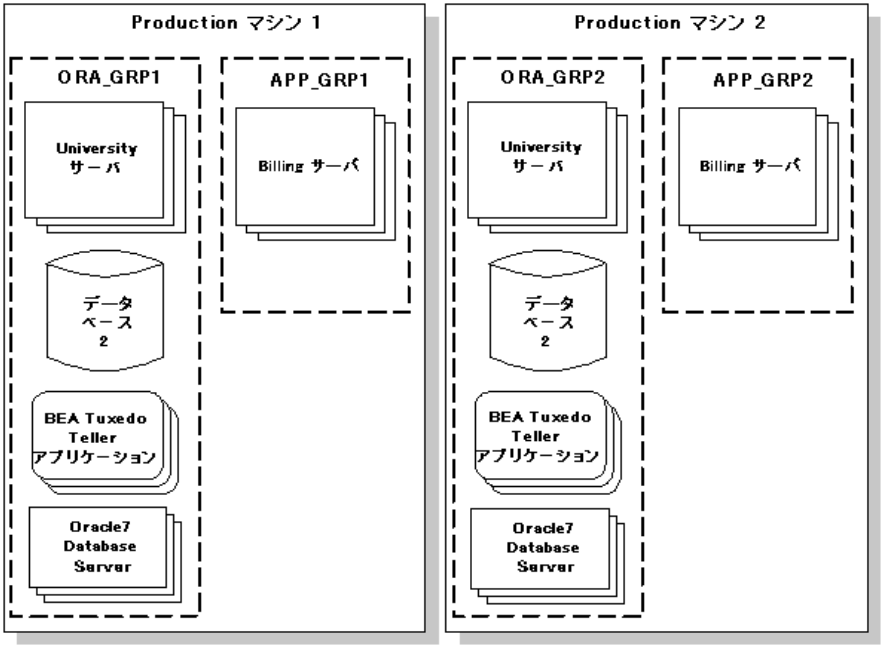


図 2-2 では、Production マシン 1 および 2 のグループの内容の違いはデータベースのみです。

- Production マシン 1 のデータベースには、100001 から 100005 までの間の ID を持つ学生の、学生情報および口座情報が格納されています。

- Production マシン 2 のデータベースには、100006 から 100010 までの間の ID を持つ学生の、学生情報および口座情報が格納されています。

注記　双方のデータベースのコース情報テーブルは同じものです。

所定のデータベース内の学生情報は、同じデータベース内の口座情報とはまったく関係がない場合があります。

Production サンプル・アプリケーションがファクトリ・ベース・ルーティングを使用して、複数のマシン間にアプリケーションの処理負荷を分散する方法の詳細については、第 2 章の 11 ページ「ファクトリ・ベース・ルーティングによるスケーリング」を参照してください。

Production アプリケーションの複製サーバ・プロセスおよびグループのコンフィギュレーション

リスト 2-1 は、Production サンプル・アプリケーションの UBBCONFIG ファイルの GROUPS および SERVERS セクションからの抜粋です。

リスト 2-1　UBBCONFIG ファイルの GROUPS および SERVERS セクション

```
*GROUPS
    APP_GRP1
        LMID       = SITE1
        GRPNO      = 2
        TMSNAME    = TMS
    APP_GRP2
        LMID       = SITE1
        GRPNO      = 3
        TMSNAME    = TMS
    ORA_GRP1
        LMID       = SITE1
        GRPNO      = 4
        OPENINFO   = "ORACLE_XA:Oracle_XA+Acc=P/scott/..."
        CLOSEINFO  = ""
        TMSNAME    = "TMS_ORA"
    ORA_GRP2
        LMID       = SITE1
        GRPNO      = 5
        OPENINFO   = "ORACLE_XA:Oracle_XA+Acc=P/scott/..."
        CLOSEINFO  = ""
        TMSNAME    = "TMS_ORA"
```

```
*SERVERS
    # デフォルトでは、各サーバのインスタンスを 2 個活性化
    # して、管理者による活性化については各サーバの
    # インスタンスを 5 個まで可能にする
    DEFAULT:
        MIN     = 2
        MAX     = 5
    tellp_server
        SRVGRP  = ORA_GRP1
        SRVID   = 10
        RESTART = N
    tellp_server
        SRVGRP  = ORA_GRP2
        SRVID   = 10
        RESTART = N

    billp_server
        SRVGRP  = APP_GRP1
        SRVID   = 10
        RESTART = N
    billp_server
        SRVGRP  = APP_GRP2
        SRVID   = 10
        RESTART = N
    univp_server
        SRVGRP  = ORA_GRP1
        SRVID   = 20
        RESTART = N
    univp_server
        SRVGRP  = ORA_GRP2
        SRVID   = 20
        RESTART = N
```

ファクトリ・ベース・ルーティングによるスケーリング

ここでは、次の内容について説明します。

- Production アプリケーションでのファクトリ・ベース・ルーティングについて
- UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション
- ファクトリでのファクトリ・ベース・ルーティングのインプリメンテーション
- 実行時の処理

このトピックでは、ファクトリ・ベース・ルーティングを使用して Production サンプル・アプリケーションをスケーリングする方法について説明します。ファクトリ・ベース・ルーティングの概要については、第 1 章の 16 ページ「ファクトリ・ベース・ルーティングの使用 (CORBA サーバのみ)」を参照してください。

Production アプリケーションでのファクトリ・ベース・ルーティングについて

この節では、Production サンプル・アプリケーションがどのようにファクトリ・ベース・ルーティングを使用するかを説明します。この機能の概要については、第 1 章の 16 ページ「ファクトリ・ベース・ルーティングの使用 (CORBA サーバのみ)」を参照してください。

ファクトリ・ベース・ルーティングを使用すると、BEA Tuxedo CORBA のロード・バランシング機能およびスケーラビリティ機能を拡張できます。Production サンプル・アプリケーションでは、ファクトリ・ベース・ルーティングを使用して、1 つのマシンに 1 つの学生のサブセットを登録する要求を、別のマシンに別の学生のサブセットを登録する要求を送信できます。アプリケーションの処理機能を向上させると、アプリケーション内でのファクトリ・ベース・ルーティングを簡単に変更して、マシンをさらに追加できます。

Production サンプル・アプリケーションでのファクトリ・ベース・ルーティングのインプリメンテーションに関する設計上の主な考慮事項は、ルーティングの基準となる値の選択です。Production サンプル・アプリケーションは、次のようにファクトリ・ベース・ルーティングを使用します。

- クライアント・アプリケーションからの Registrar オブジェクトに対する要求は、学生 ID に基づいてルーティングされます。100001 から 100005 までの学生 ID からの要求は、Production マシン 1 に送られます。学生 ID 100006 から 100010 までからの要求は、Production マシン 2 に送られます。
- Registrar オブジェクトから Teller オブジェクトへの要求は、口座番号に基づいてルーティングされます。200010 から 200014 の口座に対する課金要求は、Production マシン 1 に送られます。200015 から 200019 の口座に対する課金要求は、Production マシン 2 に送られます。

UBBCONFIG ファイルでのファクトリ・ベース・ルーティングのコンフィギュレーション

University Production サンプル・アプリケーションでは、ファクトリ・ベース・ルーティングのインプリメント方法が例示されます。ubb_b.nt コンフィギュレーション・ファイルからの INTERFACES、ROUTING、および GROUPS セクションでは、BEA Tuxedo CORBA アプリケーションでファクトリ・ベース・ルーティングをインプリメントする方法が示されます。このサンプルの ubb_p.nt または ubb_p.mk UBBCONFIG ファイルは、BEA Tuxedo ソフトウェアをインストールしたディレクトリで見つけることができます。\samples\corba\university\production サブディレクトリを参照してください。

UBBCONFIG ファイルでは、グループおよびマシンの識別方法に加えて、INTERFACES および ROUTING セクションにおいて、以下のデータを指定する必要があります。

1. INTERFACES セクションでは、ファクトリ・ベース・ルーティングを有効にするインターフェイスの名前をリストしています。各インターフェイスについて、このセクションではインターフェイスがルーティングを行う基準の種類を指定します。リスト 2-2 で示すように、ここでは識別子 FACTORYROUTING によってルーティング基準を指定します。

リスト 2-2 UBBCONFIG ファイルの INTERFACES セクション

```
INTERFACES
    "IDL:beasys.com/UniversityP/Registrar:1.0"
        FACTORYROUTING = STU_ID
    "IDL:beasys.com/BillingP/Teller:1.0"
        FACTORYROUTING = ACT_NUM
```

リスト 2-2 は、ファクトリ・ベース・ルーティングが使用されている Production サンプルの 2 つのインターフェイスの完全修飾インターフェイス名を示します。FACTORYROUTING 識別子は、ルーティング値の名前を指定します。値はそれぞれ、STU_ID および ACT_NUM です。

2. ROUTING セクションは、各ルーティング値について、表 2-1 に記載のパラメータを指定します。

表 2-1 ROUTING セクションで指定されるパラメータ

パラメータ	説明
TYPE	ルーティングのタイプを指定します。Production サンプルでは、ルーティングのタイプはファクトリ・ベース・ルーティングです。したがって、このパラメータは FACTORY と定義されます。
FIELD	ファクトリがルーティング値に挿入する変数名を指定します。Production サンプルでは、フィールド・パラメータはそれぞれ、student_id および account_number です。
FIELDTYPE	ルーティング値のデータ型を指定します。Production サンプルでは、student_id および account_number のフィールド型は long です。
RANGES	各グループにルーティングされる値を指定します。

リスト 2-3 は、Production サンプル・アプリケーションで使用される UBBCONFIG ファイルの ROUTING セクションを示します。

リスト 2-3　UBBCONFIG ファイルの ROUTING セクション

```
ROUTING
    STU_ID
        FIELD     = "student_id"
        TYPE      = FACTORY
        FIELDTYPE = LONG
       RANGES    = "100001-100005:ORA_GRP1,100006-100010:ORA_GRP2"
    ACT_NUM
        FIELD     = "account_number"
        TYPE      = FACTORY
        FIELDTYPE = LONG
       RANGES    = "200010-200014:APP_GRP1,200015-200019:APP_GRP2"
```

リスト 2-3 は、ある 1 つの範囲の ID を持つ学生に対する Registrar オブジェクト・リファレンスが、ある 1 つのサーバ・グループにルーティングされ、別の範囲の ID を持つ学生に対する Registrar オブジェクト・リファレンスが、別のグループに割り当てられることを示します。同様に、ある 1 つの範囲の口座に対する Teller オブジェクト・リファレンスは、ある 1 つのサーバ・グループにルーティングされ、別の範囲の口座に対する Teller オブジェクト・リファレンスは、別のグループにルーティングされます。

3. UBBCONFIG ファイルの ROUTING セクション内の RANGES 識別子によって指定されるグループは、識別してコンフィギュレーションする必要があります。たとえば、Production サンプルは APP_GRP1、APP_GRP2、ORA_GRP1、および ORA_GRP2 という 4 つのグループを指定します。これらのグループにはコンフィギュレーションが必要であり、これらが実行されるマシンは識別される必要があります。

   リスト 2-4 は、Production サンプルの UBBCONFIG ファイルにおける、GROUPS セクションを示します。ここで ORA_GRP1 および ORA_GRP2 の各グループがコンフィギュレーションされます。GROUPS セクション内の名前と、ROUTING セクションの RANGES パラメータで指定されたグループ名がどのように一致しているかに注目してください。これは、ファクトリ・ベース・ルーティングが正しく機能するために重要です。さらに、アプリケーションでグループをコンフィギュレーションする方法に関する何らかの変更も、ROUTING セクションに反映される必要があります。BEA Tuxedo ソフトウェアとともにパッケージされている Production サンプルは 1 台のマシンでだけ実行できるようにコンフィギュレーションされている点に注意してください。ただし、このアプリケーションを複数のマシンで実行できるようにコンフィギュレーションすることは簡単です。

リスト 2-4 UBBCONFIG ファイルの GROUPS セクション

```
*GROUPS
    APP_GRP1
        LMID      = SITE1
        GRPNO     = 2
        TMSNAME   = TMS
    APP_GRP2
        LMID      = SITE1
        GRPNO     = 3
        TMSNAME   = TMS
    ORA_GRP1
        LMID      = SITE1
        GRPNO     = 4
        OPENINFO =
"ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=.+MaxCur=5"
        CLOSEINFO = ""
        TMSNAME   = "TMS_ORA"
    ORA_GRP2
        LMID      = SITE1
        GRPNO     = 5
OPENINFO = "ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=.+MaxCur=5"
        CLOSEINFO = ""
        TMSNAME   = "TMS_ORA"
```

ファクトリでのファクトリ・ベース・ルーティングのインプリメンテーション

ファクトリでは、TP::create_object_reference() オペレーションに対する呼び出しのインプリメントと同様の方式でファクトリ・ベース・ルーティングが行われます。このオペレーションは、リスト 2-5 に示す C++ バインディングを備えています。

リスト 2-5 create_object_reference の C++ バインディング

```
CORBA::Object_ptr  TP::create_object_reference (
                       const char* interfaceName,
                       const PortableServer::oid &stroid,
                        CORBA::NVlist_ptr criteria);
```

このオペレーションに対する 3 番目のパラメータである criteria は、ファクトリ・ベース・ルーティングに使用される、名前の付いた値を指定します。ファクトリ内でファクトリ・ベース・ルーティングをインプリメントするには、NVlist をビルドする必要があります。ファクトリ・ベース・ルーティングの使用はオプションであり、この引数に依存します。ファクトリ・ベース・ルーティングを使用する代わりに、この引数に 0 (ゼロ) の値を渡すこともできます。

前述のように、Production サンプル・アプリケーションの RegistrarFactory オブジェクトは、値 STU_ID を指定します。この値は、UBBCONFIG ファイル内の次の情報に完全一致している必要があります。

- INTERFACES セクションの FACTORYROUTING 識別子で指定される、ルーティング名、タイプ、および使用できる値
- ROUTING セクションで指定される、ルーティング基準名、フィールド、およびフィールド型

RegistrarFactory オブジェクトは、リスト 2-6 に記載のコードを使用して、学生 ID を NVlist に挿入します。

リスト 2-6 RegistrarFactory オブジェクトの NVlist

```
// 学生 ID ( ルーティング基準 ) を
// CORBA NVList に挿入
CORBA::NVList_var v_criteria;
TP::orb()->create_list(1, v_criteria.out());
CORBA::Any any;
any <<= (CORBA::Long)student;
v_criteria->add_value("student_id", any, 0);
```

RegistrarFactory オブジェクトには、リスト 2-7 に示す TP::create_object_reference() オペレーションに対する呼び出しが含まれます。これにより、リスト 2-6 で作成された NVlist が渡されます。

リスト 2-7 RegistrarFactory オブジェクトでの create_object_reference の呼び出し

```
// ルーティング基準を使用して、registrar オブジェクト・
// リファレンスを作成
CORBA::Object_var v_reg_oref =
```

```
TP::create_object_reference(
    UniversityP::_tc_Registrar->id(),
    object_id,
    v_criteria.in()
);
```

Production サンプル・アプリケーションはまた、TellerFactory オブジェクトでもファクトリ・ベース・ルーティングを使用し、口座番号に基づいて Teller オブジェクトがインスタンス化されるべきグループを決定します。

実行時の処理

ファクトリでファクトリ・ベース・ルーティングをインプリメントするとき、BEA Tuxedo CORBA によってオブジェクト・リファレンスが生成されます。以下の例では、ファクトリ・ベース・ルーティングのインプリメント時に、クライアント・アプリケーションが Registrar オブジェクトへのオブジェクト・リファレンスを取得する方法を示します。

1. クライアント・アプリケーションが、RegistrarFactory オブジェクトを呼び出し、Registrar オブジェクトへのリファレンスを要求します。要求には、学生 ID が含まれています。
2. RegistrarFactory は、NVlist に学生 ID を挿入します。これは、ルーティング基準として使用されます。
3. RegistrarFactory は、TP::create_object_reference() オペレーションを呼び出し、Registrar インターフェイス名、一意の OID、および NVlist を渡します。
4. BEA Tuxedo CORBA は、ーティング・テーブルの内容を、NVlist 内の値と比較し、グループ ID を決定します。
5. BEA Tuxedo CORBA は、グループに関する情報を、オブジェクト・リファレンスに挿入します。

クライアント・アプリケーションがその後、オブジェクト・リファレンスを使用してオブジェクトを呼び出すと、BEA Tuxedo CORBA はオブジェクト・リファレンスで指定されたグループに要求をルーティングします。

注記 プロセス・エンティティの設計パターンを使用する場合は、ファクトリ・ベース・ルーティングのインプリメントは慎重に行う必要があります。オブジェクトは、グループのデータベースに格納されているエンティティしか処理できません。

設計上の追加考慮事項

ここでは、次の内容について説明します。

- 設計上の追加考慮事項について
- Registrar オブジェクトおよび Teller オブジェクトのインスタンス化
- 学生の登録が確実に正しいサーバ・グループ内で行われるようにする
- Teller オブジェクトが適切なサーバ・グループでインスタンス化されるようにする方法

設計上の追加考慮事項について

`Registrar` オブジェクトおよび `Teller` オブジェクトを設計する際には、次のことを確認してください。

- `Registrar` オブジェクトおよび `Teller` オブジェクトが、Production デプロイメント具体的には複数の複製サーバ・プロセスおよび複数のグループにわたって、適正に機能していること。University および Billing の各サーバ・プロセスが複製されている場合の設計では、これら 2 つのオブジェクトをどのようにインスタンス化するかを考慮する必要があります。
- Production BEA Tuxedo ドメインの 2 つのサーバ・グループが各々、別のデータベースを扱う場合には、所定の学生への登録および課金のオペレーションに対するクライアント要求が、正しいサーバ・グループに送られること。

オブジェクトは一意のオブジェクト ID (OID) を持ち、メソッド・バウンドである必要があります。つまり、これらのオブジェクトには、`method` 活性化方針が割り当てられている必要があります。

Registrar オブジェクトおよび Teller オブジェクトのインスタンス化

Production サンプル・アプリケーションほど高度でない University サーバ・アプリケーションでは、Registrar オブジェクトと Teller オブジェクトの実行時の振る舞いは、より単純でした。

- 各オブジェクトは、プロセス・バウンドでした。つまり、最初に呼び出されたときに活性化され、そのオブジェクトが実行されているサーバ・プロセスがシャットダウンされるまで、メモリ内に残っていました。
- BEA Tuxedo ドメインで実行されるサーバ・グループは 1 つしかなく、グループ内では University および Billing のサーバ・プロセスは 1 つしかなかったため、すべてのクライアント要求は、同じオブジェクトに転送されました。複数のクライアント要求が BEA Tuxedo ドメインに到達する一方で、これらの各オブジェクトが処理するクライアントは一度に 1 つずつでした。
- 各オブジェクトのインスタンスは、それらが実行されているサーバ・プロセスに 1 つしかなかったため、どちらのオブジェクトにも一意の OID は不要でした。各オブジェクトの OID が指定するのは、インターフェイス・リポジトリ ID のみでした。

しかし今では、University および Billing サーバ・プロセスは複製されているため、BEA Tuxedo CORBA は Registrar オブジェクトと Teller オブジェクトの複数インスタンスを区別できなくてはなりません。たとえば、グループ内で 2 つの University サーバ・プロセスが実行されている場合、BEA Tuxedo CORBA には、1 つ目の University サーバ・プロセスで実行されている Registrar オブジェクトと、2 つ目のサーバ・プロセスで実行されている Registrar オブジェクトを区別する手段が必要です。これらのオブジェクトの複数のインスタンスを区別するには、各オブジェクト・インスタンスが一意的である必要があります。

各 Registrar および Teller オブジェクトを一意のものにするには、これらのオブジェクトのファクトリで、これらに対するオブジェクト・リファレンスを作成する方法を変更しなければなりません。たとえば Basic サンプル・オブジェクトでは、RegistrarFactory オブジェクトが Registrar オブジェクトへのオブジェクト・リファレンスを作成すると、TP::create_object_reference() オペレーションが、文字列 registrar のみからなる OID を指定していました。しか

し、Production サンプル・アプリケーションでは、同じ TP::create_object_reference() オペレーションが、生成された一意の OID を使用します。

各 Registrar および Teller オブジェクトに一意の OID を付与した結果、BEA Tuxedo ドメインではこれらのオブジェクトの複数のインスタンスが、同時に実行可能です。この特性は、状態を持たないオブジェクト・モデルでは一般的なものであり、BEA Tuxedo ドメインが、高い性能を提供しつつ、高度にスケーラブルであり得ることの一例です。

最後に、一意の Registrar および Teller オブジェクトは、それらに対するクライアント要求があるたびにメモリにロードする必要があるため、呼び出しが完了したときにはこれらのオブジェクトを非活性化し、関連するオブジェクト状態がアイドル状態のままメモリに残存しないようにすることが重要です。Production サーバ・アプリケーションは、インプリメンテーション・コンフィギュレーション・ファイル (ICF) 内のこれら 2 つのオブジェクトに、method 活性化方針を割り当てることによって、この問題に対処します。

学生の登録が確実に正しいサーバ・グループ内で行われるようにする

複製サーバ・グループを使用する、主なスケーラビリティ上の利点は、複数台のマシンに処理を分散できることです。しかし、University サンプル・アプリケーションの場合のように、アプリケーションがデータベースと対話する場合は、これら複数のサーバ・グループがデータベースとの対話に与える影響を考慮することが重要です。

多くの場合、デプロイされているマシンに、データベースは 1 つずつ関連付けられています。サーバ・アプリケーションが複数台のマシンに分散されている場合は、データベースをどのようにセットアップするかを検討する必要があります。

この章で説明しているように、Production サンプル・アプリケーションでは、2 つのデータベースを使用します。しかし、このアプリケーションは、簡単にそれ以上のデータベースに対応できるようコンフィギュレーションできます。使用するデータベースの数は、システム管理者が決定できます。

Production サンプル・アプリケーションでは、学生および口座の情報は、2 つのデータベース間に分割されますが、コース情報は同一です。コース情報は、コース登録用途では読み取り専用となっているため、両方のデータベースのコース情報が同一であることは問題にはなりません。一方で、学生情報および口座情報に

ついては読み書きが行われます。複数のデータベースが学生および口座についても同一のデータを格納するとしたら (つまり、データベースが分割されていない場合)、アプリケーションでは、学生情報または口座情報が変更されるたびにデータベース全体にわたって学生および口座の情報の更新を同期する処理のオーバーヘッドに対処する必要が生じるでしょう。

Production サンプル・アプリケーションは、ファクトリ・ベース・ルーティングによって、1 台のマシンに要求の 1 セットを、別のマシンに要求の別の 1 セットを送信します。RegistrarFactory オブジェクトでファクトリ・ベース・ルーティングがどのようにインプリメントされるかは、Registrar オブジェクトへのリファレンスがどのように作成されるかによって変わります。

たとえば、クライアント・アプリケーションが RegistrarFactory オブジェクトに要求を送信して、Registrar オブジェクトのオブジェクト・リファレンスを取得した場合、クライアント・アプリケーションはその要求に学生 ID を含めます。クライアント・アプリケーションは、RegistrarFactory オブジェクトが返すオブジェクト・リファレンスを使用して、その後の Registrar オブジェクトへのすべての呼び出しを、特定の学生のために行う必要があります。ファクトリによって返されたオブジェクト・リファレンスは、グループ固有のものだからです。したがって、たとえばクライアント・アプリケーションがその後、Registrar オブジェクトに対して get_student_details() オペレーションを呼び出すと、クライアント・アプリケーションでは、Registrar オブジェクトが、その学生のデータを格納しているデータベースと関連付けられたサーバ・グループにおいて、確実に活性化されます。

この機能を示すために、Production サンプル・アプリケーションにインプリメントされている以下の実行シナリオを検討してみます。

1. クライアント・アプリケーションは、RegistrarFactory オブジェクトの find_registrar() オペレーションを呼び出します。この呼び出しには、学生 ID 1000003 が含まれます。
2. BEA Tuxedo CORBA は、クライアント要求を任意の RegistrarFactory オブジェクトにルーティングします。
3. RegistrarFactory オブジェクトは、学生 ID を使用し、UBBCONFIG ファイルのルーティング情報に基づいて、ORA_GRP1 内の Registrar オブジェクトへのオブジェクト・リファレンスを作成し、そのオブジェクト・リファレンスをクライアント・アプリケーションに返します。
4. クライアント・アプリケーションは、Registrar オブジェクトの register_for_courses() オペレーションを呼び出します。

5. BEA Tuxedo CORBA はクライアント要求を受け取り、それをオブジェクト・リファレンスで指定されたサーバ・グループにルーティングします。この場合、クライアント要求は、Production マシン 1 の ORA_GRP1 内の University サーバ・プロセスに送られます。
6. University サーバ・プロセスは、Registrar オブジェクトをインスタンス化し、それにクライアント呼び出しを送信します。

上記の説明の RegistrarFactory オブジェクトは、クライアント・アプリケーションに、ORA_GRP1 内でのみインスタンス化が可能な Registrar オブジェクトへの一意のリファレンスを返します。このグループには Production マシン 1 上で実行され、100001 から 100005 までの ID を持つ学生の学生データを格納したデータベースがあります。したがって、クライアント・アプリケーションが所定の学生のために、その後の要求をこの Registrar オブジェクトに送信すると、Registrar オブジェクトは適正なデータベースと対話を行います。

Teller オブジェクトが適切なサーバ・グループでインスタンス化されるようにする方法

Registrar オブジェクトは Teller オブジェクトが必要になると、University サーバ・オブジェクトにキャッシュされていた TellerFactory オブジェクト・リファレンスを使用して、TellerFactory オブジェクトを呼び出します。

しかし、TellerFactory オブジェクトではファクトリ・ベース・ルーティングが使用されているため、Registrar オブジェクトは Teller オブジェクトへのリファレンス要求時に、学生の口座番号を渡します。このようにして TellerFactory オブジェクトは、正しいデータベースを備えたグループ内の Teller オブジェクトへのリファレンスを作成します。

注記 Production サンプル・アプリケーションが適正に動作するためには、システム管理者がサーバ・グループとデータベースのコンフィギュレーションを正しく行うことが不可欠です。具体的には、システム管理者はルーティング・テーブルで指定されたルーティング基準と、これらの基準を使用した要求のルーティング先となるデータベースを、確実に一致させる必要があります。Production サンプルを例にすると、指定されたグループにあるデータベースは、そのグループにルーティングされた要求に正しく対応する学生情報および口座情報を含んでいる必要があります。

アプリケーションをさらにスケーリングする方法

将来的には、Production サンプル・アプリケーションのシステム管理者が BEA Tuxedo ドメインの容量を増やす必要を感じる場合もあります。たとえば、今後大学の学生数が著しく増えたり、いくつかのキャンパスを包含する、州全体の大学システムのコース登録プロセスに対応するように Production アプリケーションが拡張されたりすることが考えられます。これを行うのに、アプリケーションを修正したり再ビルドしたりする必要はありません。

システム管理者は、以下の処理によって、容量を追加し続けることができます。

- Production サンプル・アプリケーションのサーバ・グループを、追加のマシンにわたるように複製します。

  システム管理者は、UBBCONFIG ファイルを修正し、追加のサーバ・グループ、これらのグループで実行されるサーバ・プロセス、およびサーバ・グループが動作するマシンを指定する必要があります。

- ファクトリ・ベース・ルーティング・テーブルを変更します。

  たとえば、Production サンプル・アプリケーションの既存の 2 つのグループへルーティングする代わりに、システム管理者は UBBCONFIG ファイルのルーティング規則を変更して、BEA Tuxedo ドメインに追加されたサーバ・グループにわたるよう、アプリケーションをさらに分割することができます。ルーティング・テーブルへの変更はすべて、UBBCONFIG ファイル内のコンフィギュレーション済みのサーバ・グループおよびマシンの情報と一致している必要があります。

注記 データベースを使用する既存の BEA Tuxedo CORBA アプリケーションに容量を追加する場合、特にファクトリ・ベース・ルーティングを使用しているときは、データベースのセットアップに対する影響も考慮する必要があります。たとえば、Production サンプル・アプリケーションが 6 台のマシンに分散されている場合、各マシン上のデータベースを適切かつ UBBCONFIG ファイルのルーティング・テーブルに従ってセットアップする必要があります。

3　CORBA アプリケーションの分散

ここでは、次の内容について説明します。

- アプリケーションを分散する理由
- データ依存型ルーティングの使用 (BEA Tuxedo ATMI サーバのみ)
- UBBCONFIG ファイルのコンフィギュレーション
- factory_finder.ini のコンフィギュレーション (CORBA アプリケーションのみ)
- ルーティングをサポートするためのドメイン・ゲートウェイ・コンフィギュレーション・ファイルの変更

このトピックでは、CORBA アプリケーションを例に、BEA Tuxedo CORBA 環境でのアプリケーションの分散方法を説明します。

アプリケーションを分散する理由

ここでは、次の内容について説明します。

- アプリケーションの分散について
- 分散アプリケーションの利点
- アプリケーション分散の特性

アプリケーションの分散について

アプリケーションを分散すると、アプリケーションのある部分を選択して論理的にグループ化し、そのグループを実行する場所を選択できます。アプリケーションの分散は、UBBCONFIG ファイルの GROUPS セクションに複数のエントリを作成し、アプリケーションの資源またはタスクをグループ間に分割することによって行います。サーバのグループを作成することにより、非常に大規模なアプリケーションをコンポーネント・ビジネス・アプリケーションに分割し、これらの各論理コンポーネントを管理可能なサイズで最適な場所に配置できます。

分散アプリケーションの利点

分散アプリケーションの利点には、以下のようなものがあります。

- スケーラビリティ ー アプリケーションで対処できる負荷を増やすため、以下のことを行います。
  - グループ内に追加のサーバ・プロセスをもたらします。
  - アプリケーションにマシンを追加し、マシン間にグループを再分散します。
  - アプリケーション内のほかのマシンに対してグループを複製し、ロード・バランシングを行います。
  - データベースをセグメントに分割し、これらの個別のデータベース・セグメントを扱うグループに到達するデータ依存型ルーティングを使用します (BEA Tuxedo ATMI システム)。

BEA Tuxedo CORBA システムでは、ファクトリ・ベース・ルーティングを使用し、複数のサーバ・グループ、および必要であれば複数のマシンにわたって、特定の CORBA インターフェイスの処理を分散できます。この機能を使うと処理負荷が分散できるので、同時実行されるリソース集中型のアプリケーションが、利用可能な CPU、メモリ、ディスク I/O、およびネットワーク資源をめぐって競合した場合に発生する、処理上のボトルネックを回避できます。ファクトリ・ベース・ルーティングの使用例については、第 2 章の 11 ページ「ファクトリ・ベース・ルーティングによるスケーリング」を参照してください。

BEA Tuxedo CORBA のスケーラビリティの特長の詳細については、「第 1 章 BEA Tuxedo CORBA アプリケーションのスケーリング」を参照してください。

- 開発と保守の容易性 ― ビジネス・アプリケーション・ロジックを適切に定義されたメッセージまたはインターフェイスを介して通信するサービスまたはコンポーネントに分割することにより、開発と保守の双方を、同様に分割して、簡略化できます。
- 信頼性 ― 複数のマシンを使用中にそのうちの 1 つが停止しても、それ以外のマシンは動作を続行できます。同様に、グループ内に複数のサーバ・プロセスがあってそのうちの 1 つが失敗しても、それ以外のプロセスを使用して作業を行えます。最後に、マシンが停止しても、アプリケーション内に複数のマシンがあれば、それらのほかのマシンを使用して負荷を処理できます。
- 自律的なアクションの調整 ― アプリケーションが個別になっていれば、アプリケーション間で、自律的なアクションを 1 つの論理的な作業単位として調整できます。自律的なアクションとは、複数のサーバ・グループと複数のリソース・マネージャ・インターフェイスが関与するアクションです。

アプリケーション分散の特性

分散アプリケーションには、次の特性があります。

- クライアント / サーバ型のモデルを拡張できます。
- 複数のサーバ・グループを確立できます。
- BEA Tuxedo サービスまたは BEA Tuxedo CORBA インターフェイスに透過的にアクセスできます。
- BEA Tuxedo では、データ依存型のデータ分離を行えます。

- BEA Tuxedo CORBA では、CORBA オブジェクトを複数グループまたは複数マシンに分割したり、アプリケーション・ファクトリ・インターフェイスおよびアプリケーション・インターフェイスを分散したりできます。
- 複数リソースを管理できます。
- ネットワーク化されたモデルをサポートしています。

データ依存型ルーティングの使用 (BEA Tuxedo ATMI サーバのみ)

ここでは、次の内容について説明します。

- データ依存型ルーティングについて
- データ依存型ルーティングの特性
- 分散アプリケーションの例
- 分散アプリケーションにおける UBBCONFIG セクションの例

注記　このトピックは、BEA Tuxedo にのみ適用されます。

データ依存型ルーティングについて

データ依存型ルーティングは、サービス要求が、送信されたバッファ内のデータ値に基づき、クライアント (またはクライアントとして動作しているサーバ) によって、特定グループ内のサーバにルーティングされるメカニズムです。サービス呼び出しの内部コードにおいて、BEA Tuxedo はデータ・フィールドと掲示板共用メモリ内で探し出したルーティング基準を比較して、送信先となるサーバを選択します。

所定のどのサービスについても、UBBCONFIG ファイルの SERVICES セクションでルーティング基準識別子を指定できます。ルーティング基準識別子 (特にサーバ・グループに対するデータ範囲のマッピング) は、ROUTING セクションで指定します。

データ依存型ルーティングの特性

データ依存型ルーティングの特性は次のとおりです。

- グループ内のサーバに割り当てられたサービス要求は、データ値に基づいています。
- ルーティングは、掲示板の基準を使用し、サーバ呼び出し中に発生します。
- サービスに対するルーティング基準識別子は、UBBCONFIG ファイルの SERVICES セクションで指定されます。
- ルーティング基準識別子は、UBBCONFIG ファイルの UBBCONFIG セクションで定義されます。

分散アプリケーションの例

表 3-1 は、クライアント要求がサーバにルーティングされるしくみを説明します。この例では、bankapp と呼ばれる銀行取引アプリケーションでデータ依存型ルーティングが使用されています。bankapp には、3 つのグループ (BANKB1、BANKB2、および BANKB3) と、2 つのルーティング基準 (Account_ID および Branch_ID) があります。サービス WITHDRAW、DEPOSIT、および INQUIRY は、Account_ID フィールドを使用してルーティングされます。サービス OPEN および CLOSE は、Branch_ID フィールドを使用してルーティングされます。

表 3-1 サンプル分散アプリケーションのデータ依存型ルーティング

サーバ・グループ	ルーティング基準	サービス
BANKB1	Account_ID: 10000 - 49999	WITHDRAW、DEPOSIT、および INQUIRY
	Branch_ID: 1 - 4	OPEN および CLOSE
BANKB2	Account_ID: 50000 - 79999	WITHDRAW、DEPOSIT、および INQUIRY
	Branch_ID: 5 - 7	OPEN および CLOSE

表 3-1 サンプル分散アプリケーションのデータ依存型ルーティング (続き)

サーバ・グループ	ルーティング基準	サービス
BANKB3	Account_ID: 80000 - 109999	WITHDRAW、DEPOSIT、および INQUIRY
	Branch_ID: 8 - 10	OPEN および CLOSE

分散アプリケーションにおける UBBCONFIG セクションの例

リスト 3-1 は、BEA Tuxedo システムでデータ依存型ルーティングを実現するためのコンフィギュレーション・ファイルにおける、GROUPS、SERVICES、および ROUTING の各セクションを含む UBBCONFIG ファイルの例を示します。

リスト 3-1 UBBCONFIG ファイルの例

```
*GROUPS
BANKB1              GRPNO=1
BANKB2              GRPNO=2
BANKB3              GRPNO=3
#
*SERVICES
WITHDRAW            ROUTING=ACCOUNT_ID
DEPOSIT             ROUTING=ACCOUNT_ID
INQUIRY             ROUTING=ACCOUNT_ID
OPEN_ACCT           ROUTING=BRANCH_ID
CLOSE_ACCT          ROUTING=BRANCH_ID
#
*ROUTING
ACCOUNT_ID          FIELD=ACCOUNT_ID BUFTYPE="FML"
                     RANGES="MIN - 9999:*,
                        10000-49999:BANKB1,
                        50000-79999:BANKB2,
                        80000-109999:BANKB3,
                        *:*"
BRANCH_ID           FIELD=BRANCH_ID BUFTYPE="FML"
                     RANGES="MIN - 0:*,
                        1-4:BANKB1,
                        5-7:BANKB2,
                        8-10:BANKB3,
                          *:*"
```

UBBCONFIG ファイルのコンフィギュレーション

ここでは、次の内容について説明します。

- 分散アプリケーションにおける UBBCONFIG ファイルについて
- GROUPS セクションの変更
- SERVICES セクションの変更
- INTERFACES セクションの変更
- ROUTING セクションの作成

UBBCONFIG ファイルの詳細については、『BEA Tuxedo アプリケーションの設定』の「コンフィギュレーション ファイルの作成」を参照してください。

分散アプリケーションにおける UBBCONFIG ファイルについて

UBBCONFIG ファイルには、以下のようなデータ依存型ルーティング (BEA Tuxedo) またはファクトリ・ベース・ルーティング (BEA Tuxedo CORBA) の説明が含まれます。

- GROUPS セクションには、システムを分散するために必要な数のサーバ・グループが挿入されます。これにより、システムは要求を特定グループのサーバにルーティングできるようになります。これらのグループは、すべて同じサイトに設定することも (SHM モード)、ネットワーク化がなされている場合に複数のサイトに分散することも (MP モード) できます。
- BEA Tuxedo でデータ依存型ルーティングを行うには、SERVICES セクションに、ROUTING パラメータを使用する各サービスのルーティング基準がリストされている必要があります。

  注記 サービスに複数のエントリがあり、それぞれが異なった SRVGRP パラメータを備えている場合、すべてのエントリで、サービスに対する一貫性を確保するために、ROUTING の設定を同じにしなければなりませ

ん。1 つのサービスは、1 つのフィールドにしかルーティングできず、このフィールドは、同じサービスでは同じものにする必要があります。

- BEA Tuxedo CORBA でファクトリ・ベース・ルーティングを行うには、INTERFACES セクションに、FACTORYROUTING パラメータを使用する各 CORBA インターフェイスのルーティング基準名がリストされている必要があります。このパラメータは、ROUTING セクションで定義されたルーティング基準名に設定されます。
- コンフィギュレーション・ファイルに ROUTING セクションを追加して、システムが要求を特定グループのサーバに送信できるように、データ範囲とグループ間のマッピングを示します。ROUTING セクションの各項目には、INTERFACES セクションで使用される識別子 (BEA Tuxedo ATMI の場合) または SERVICES セクションで使用される識別子 (BEA Tuxedo の場合) が含まれます。

GROUPS セクションの変更

GROUPS セクションのパラメータは、分散トランザクション処理に関する次の 2 つの重要な側面をインプリメントします。

- これらのパラメータにより、サーバ・グループが特定の LMID およびリソース・マネージャの特定のインスタンスに関連付けられます。
- サーバ・グループに対して、代替マシンを示す別の LMID を関連付けることにより、サーバ・グループを代替マシンに移行できます (ただし、MIGRATE オプションが指定されている場合)。

表 3-2 では、GROUPS セクションのパラメータを説明します。

表 3-2 GROUPS セクションで指定されるパラメータ

パラメータ	説明
LMID	このサーバ・グループがこの特定のマシン上で実行されることを示すには、LMID を MACHINES セクションに割り当てる必要があります。値の後に、カンマで区切って別の LMID を指定し、代替マシンを設定することもできます。MIGRATE オプションが指定されている場合は、サーバ・グループをこの代替マシンに移行できます。グループ内のサーバを移行するには、RESTART=Y を指定する必要があります。
GRPNO	このサーバ・グループにグループ番号を示す数値を関連付けます。0 より大きく、30000 未満の番号を指定する必要があります。番号は、このコンフィギュレーション・ファイルの GROUPS セクションのエントリの中で一意でなければなりません (必須)。
TMSNAME	このサーバ・グループに関連付けるトランザクション管理サーバ (TMS) を指定します。
TMSCOUNT	このサーバ・グループ用に起動する TMSNAME のコピー数を指定します。最小値は 2 です。値を指定しない場合は、デフォルトの 3 が指定されます。サーバ・グループ用に起動された TMSNAME サーバはすべて、MSSQ セットで自動的に設定されます (省略可能)。

表 3-2 GROUPS セクションで指定されるパラメータ (続き)

パラメータ	説明
OPENINFO	特定のリソース・マネージャの特定のインスタンスを開くために必要な情報を指定するか、またはそのような情報がこのサーバ・グループには不要であることを示します。リソース・マネージャを OPENINFO パラメータで指定した場合、データベース名やアクセス・モードなどの情報が含まれます。値の文字列全体を 256 文字以下とし、二重引用符で囲む必要があります。OPENINFO 文字列の形式は、基となるリソース・マネージャのベンダごとに異なります。ベンダ固有の文字列の先頭は、rm_name: となります。これはベンダのトランザクション・インターフェイス (XA インターフェイス) の公開名の直後にコロン (:) を付けたものです。 TMSNAME が設定されていないか、TMS に設定されている場合、OPENINFO パラメータは無視されます。TMSNAME が設定されているが、OPENINFO 文字列の設定がヌル文字列 ("") になっている場合、またはこのパラメータがエントリ内に登場しない場合は、グループのリソース・マネージャが存在しているが、open オペレーションを実行するための情報を必要としていないことを意味します。
CLOSEINFO	データベースを閉じるときにリソース・マネージャが必要とする情報を指定します。このパラメータは省略することも、ヌル文字列を指定することもできます。デフォルト値はヌル文字列です。

SERVICES セクションの変更

SERVICES セクションのパラメータは、アプリケーション・サービスの処理方法を制御します。このセクションのエントリの行は、識別子名によってサービスと関連付けられます。SRVGRP パラメータは、1 つのサービスを複数のサーバにリンクするために用意されています。このパラメータは、サービスのインスタンスを示すパラメータを特定のサーバ・グループに関連付けます。

変更するパラメータ

SERVICES セクションには、特に BEA Tuxedo ATMI サービスを使用する BEA Tuxedo CORBA アプリケーションの分散トランザクション処理 (DTP) に関連付けられるパラメータが 2 つあります。AUTOTRAN および TRANTIME です。

表 3-4 では、SERVICES セクションのパラメータを説明します。

表 3-3 SERVICES セクションで指定されるパラメータ

パラメータ	説明
AUTOTRAN	このサービスで受信されたメッセージが既にトランザクション・モードではない場合に、自動的にトランザクションを開始するかどうかを判断します。デフォルト値は、N です。このパラメータの使用については、アプリケーションのサービスをコーディングするプログラマとの間で調整を行ってください。
TRANTIME	このサービスで自動的に開始されたトランザクションの、タイムアウト値を秒単位で指定します。デフォルト値は 30 秒です。これを指定するのは、AUTOTRAN=Y や、ほかのタイムアウト値が必要な場合のみです。

SERVICES セクションの例

リスト 3-2 では、SERVICES セクションの例を示します。

リスト 3-2 Production サンプルの SERVICES セクション

```
*SERVICES

    # Tuxedo Teller アプリケーション・サービスを公開する
    #
    DEBIT
        AUTOTRAN=Y
    CREDIT
        AUTOTRAN=Y
    CURRBALANCE
        AUTOTRAN=Y
```

INTERFACES セクションの変更

INTERFACES セクションのパラメータは、アプリケーション・インターフェイスの処理方法を制御します。このセクションのエントリの行は、識別子名によってインターフェイスと関連付けられます。SRVGRP パラメータは、1 つのインターフェイスを複数のサーバにリンクするために用意されています。このパラメータは、インターフェイスのインスタンスを示すパラメータを特定のサーバ・グループに関連付けます。

変更するパラメータ

INTERFACES セクションには、特に分散トランザクション処理 (DTP) に関連付けられるパラメータが 3 つあります。FACTORYROUTING、AUTOTRAN、および TRANTIME です。

表 3-4 では、INTERFACES セクションのパラメータを説明します。

表 3-4 INTERFACES セクションで指定されるパラメータ

パラメータ	説明
FACTORYROUTING = criterion-name	この BEA Tuxedo CORBA インターフェイスのファクトリ・ベース・ルーティングに使用されるルーティング基準名を指定します。ファクトリ・ベース・ルーティングを要求するインターフェイスに対しては、FACTORYROUTING パラメータを指定する必要があります。
AUTOTRAN	このインターフェイスで受信されたメッセージが既にトランザクション・モードではない場合に、自動的にトランザクションを開始するかどうかを判断します。デフォルト値は、N です。このパラメータの使用については、アプリケーションの ICF ファイルの transaction policy オプションの設定とパラメータが一致するように、アプリケーションのインターフェイスをコーディングするプログラマとの間で調整を行ってください。
TRANTIME	このインターフェイスで自動的に開始されたトランザクションの、タイムアウト値を秒単位で指定します。デフォルト値は 30 秒です。これを指定するのは、AUTOTRAN=Y およびデフォルト以外のタイムアウト値が必要な場合のみです。

表 3-4 INTERFACES セクションで指定されるパラメータ (続き)

パラメータ	説明
LOAD = number	CORBA インターフェイスがシステムに与えると予想される相対的な負荷を表す、1 から 100 の間の任意の数値を指定します。数の設定スキーマは、このアプリケーションで使用されるほかの CORBA インターフェイスに割り当てられた LOAD 数値を基準としています。デフォルト値は 50 です。BEA Tuxedo システムは、要求のルーティング先として最適なサーバを選択するためにこの数字を使用します。

INTERFACES セクションの例

リスト 3-2 では、INTERFACES セクションの例を示します。

リスト 3-3 INTERFACES セクションの例

```
*INTERFACES

   "IDL:beasys.com/UniversityP/Registrar:1.0"
        FACTORYROUTING = STU_ID
        AUTOTRAN=Y
        TRANTIME=50

   "IDL:beasys.com/BillingP/Teller:1.0"
        FACTORYROUTING = ACT_NUM
        AUTOTRAN=Y
```

ROUTING セクションの作成

BEA Tuxedo のデータ依存型ルーティングまたは BEA Tuxedo CORBA のファクトリ・ベース・ルーティングをサポートする ROUTING パラメータについては、『BEA Tuxedo アプリケーションの設定』の「コンフィギュレーション ファイルの作成」を参照してください。

リスト 3-4 は、Production サンプル・アプリケーションでファクトリ・ベース・ルーティングに使用される UBBCONFIG ファイルの ROUTING セクションを示します。

リスト 3-4 Production サンプルの ROUTING セクション

```
*ROUTING

   STU_ID
      FIELD     = "student_id"
      TYPE      = FACTORY
      FIELDTYPE = LONG
      RANGES    = "100001-100005:ORA_GRP1,100006-100010:ORA_GRP2"

   ACT_NUM
      FIELD     = "account_number"
      TYPE      = FACTORY
      FIELDTYPE = LONG
      RANGES    = "200010-200014:APP_GRP1,200015-200019:APP_GRP2"
```

factory_finder.ini のコンフィギュレーション (CORBA アプリケーションのみ)

CORBA アプリケーションの場合、複数ドメインにわたってファクトリ・ベース・ルーティングをコンフィギュレーションするには、現在の (ローカル) ドメインで使用されているが、別の (リモート) ドメインに存在するファクトリ・オブジェクトを識別するために、factory_finder.ini ファイルをコンフィギュレーションする必要があります。詳細については、『BEA Tuxedo Domains コンポーネント』の「複数の CORBA ドメインを設定する」を参照してください。

ルーティングをサポートするためのドメイン・ゲートウェイ・コンフィギュレーション・ファイルの変更

ここでは、次の内容について説明します。

- ドメイン・ゲートウェイ・コンフィギュレーション・ファイルについて
- DMCONFIG ファイルの DM_ROUTING セクションのパラメータ (BEA Tuxedo ATMI のみ)

この節は BEA Tuxedo のみを対象とし、ルーティングをサポートするためにドメイン・ゲートウェイ・コンフィギュレーションを変更する方法と、この処理が必要である理由を説明します。ドメイン・ゲートウェイ・コンフィギュレーション・ファイルの詳細については、『BEA Tuxedo Domains コンポーネント』の「複数の CORBA ドメインを設定する」を参照してください。

ドメイン・ゲートウェイ・コンフィギュレーション・ファイルについて

ドメイン・ゲートウェイ・コンフィギュレーションに関する情報は、バイナリ形式の BDMCONFIG ファイルに格納されています。DMCONFIG ファイル (ASCII) は、任意のテキスト・エディタで作成および編集できます。コンパイル済みの BDMCONFIG ファイルは、実行中のシステムで dmadmin(1) コマンドを使用して更新できます。

Domains 機能を必要とする各 BEA Tuxedo アプリケーションには、BDMCONFIG ファイルを 1 つ作成する必要があります。BDMCONFIG ファイルへのアクセスは、Domains 管理サーバである DMADM(5) から行います。ゲートウェイ・グループが起動されると、ゲートウェイ管理サーバである GWADM(5) は、そのグループに必要なコンフィギュレーションのコピーを DMADM サーバに要求します。また、GWADM サーバと DMADM サーバは、コンフィギュレーションにおける実行時の変更が、対応するドメイン・ゲートウェイ・グループに反映されるようにします。

注記 DMCONFIG ファイルの変更の詳細については、『BEA Tuxedo Domains コンポーネント』の「複数の CORBA ドメインを設定する」を参照してください。

DMCONFIG ファイルの DM_ROUTING セクションのパラメータ (BEA Tuxedo ATMI のみ)

DM_ROUTING セクションでは、型付きバッファである FML、VIEW、X_C_TYPE、および X_COMMON を使用したサービス要求のデータ依存型ルーティングに関する情報を提供します。DM_ROUTING セクション内にある各行の形式は、CRITERION_NAME です。ここで、CRITERION_NAME は SERVICES セクションで指定されたルーティング・エントリの名前 (識別子名) です。CRITERION_NAME エントリの長さは 15 文字以下です。

指定するパラメータ

表 3-5 では、DM_ROUTING セクションのパラメータを説明します。

表 3-5 DM_ROUTING セクションで指定されるパラメータ

パラメータ	説明
FIELD = identifier	ルーティング・フィールドの名前を指定します。長さは 30 文字以下です。このフィールドは、FML フィールド・テーブルで識別されたフィールド名 (FML バッファの場合)、または FML VIEW テーブルで識別されたフィールド名 (VIEW、X_C_TYPE、または X_COMMON バッファの場合) と見なされます。FLDTBLDIR 環境変数および FIELDTBLS 環境変数は FML フィールド・テーブルを見つけるために使用され、VIEWDIR 環境変数および VIEWFILES 環境変数は、FML VIEW テーブルを見つけるために使用されます。FML32 バッファ内のフィールドがルーティングに使用される場合は、8191 以下の数値をフィールド番号として指定します。

表 3-5 DM_ROUTING セクションで指定されるパラメータ (続き)

パラメータ	説明
BUFTYPE = "type1[:subtype1[,subtype2 . . .]][;type2[:subtype3[, . . .]]] . . ."	このルーティング・エントリで有効なデータ・バッファのタイプとサブタイプのリストを指定します。有効なタイプは、FML、VIEW、X_C_TYPE、および X_COMMON に限定されます。 タイプ FML ではサブタイプは指定されません。それ以外のタイプにはサブタイプが必要です。ただし * は使用できません。 タイプとサブタイプのペアのうち、重複するものは同じルーティング基準名として指定できません。タイプとサブタイプのペアが一意の場合、複数のルーティング・エントリは同じ基準名を持つことができます。これは必須パラメータです。 単一のルーティング・エントリに複数のバッファ・タイプが指定される場合、各バッファ・タイプに対するルーティング・フィールドのデータ型は同じでなければなりません。フィールド値が設定されていないか (FML バッファの場合)、または特定の範囲と一致しておらず、ワイルドカードの範囲が指定されていない場合、リモート・サービスの実行を要求したアプリケーション・プロセスに対してエラーが返されます。

表 3-5 DM_ROUTING セクションで指定されるパラメータ (続き)

パラメータ	説明
RANGES ="range1:rdom1[,range2:rdom2 ...]"	ルーティング・フィールドの範囲と関連付けられたリモート・ドメイン名 (RDOM) を指定します。文字列は二重引用符で囲み、range/RDOM のペアをカンマで区切って順番に並べた形式にします。 範囲は、単一の値 (符号付き数値または一重引用符で囲んだ文字列)、または、*lower - upper* (lower と upper は共に符号付き数値または一重引用符で囲んだ文字列) の形式で表します。*lower* には、*upper* 以下の値を設定します。文字列値に一重引用符を埋め込むには、一重引用符の前にバックスラッシュを 2 つ挿入します。たとえば、「O'Brien」は「O\\'Brien」になります。 ■ 関連する FIELD のデータ型の最小値を示すには、MIN を使用します。文字列と carray の最小値にはヌル文字列を指定します。文字フィールドの最小値には、0 を指定します。数値の場合、これはフィールドに格納できる最小値です。 ■ 関連する FIELD のデータ型の最大値を示すには、MAX を使用します。文字列と carray の最大値には、8 進数値の 255 文字の無限文字列を指定します。文字フィールドの最大値には、単一の 8 進数値の 255 文字を指定します。数値の場合は、数値としてフィールドに格納できる最大値です。 したがって、MIN - -5 は -5 以下のすべての数を表し、6 - MAX は 6 以上のすべての数を表します。 範囲 (range) 内のメタキャラクタ * (ワイルドカード) は、既にエントリとして指定した範囲では使用されなかった任意の値を示します。各エントリでは 1 つのワイルドカードによる範囲指定だけが可能です。* は最後に指定します。続けて範囲を指定すると無視されます。

ルーティング・フィールドの説明

ルーティング・フィールドには、FML または VIEW でサポートされている任意のデータ型を指定できます。数値のルーティング・フィールドには、数値による範囲値、文字列のルーティング・フィールドには文字列による範囲値を指定します。

文字列、carray、および文字フィールド型の文字列の範囲値は、1 組の一重引用符で囲みます。前に符号を付けることはできません。short 型および long 型の整数値は数値の文字列であり、オプションで先頭に正の符号 (+) または負の符号 (-) を指定できます。浮動小数点数は、C コンパイラまたは atof() で受け付けられる形式です。つまり、符号 (オプション)、数字の文字列 (オプションで小数点を追加)、e または E (オプション)、符号または空白文字 (オプション)、整数という形式で指定します。

フィールド値が範囲と一致するときに、関連する RDOM 値には、要求がルーティングされるリモート・ドメインを指定します。RDOM 値に * を指定すると、ゲートウェイ・グループが認識する任意のリモート・ドメインに要求が送信されることを示します。range/RDOM の組み合わせでは、範囲と RDOM はコロン (:) で区切られます。

ルーティングを使用した 5 サイトのドメイン・コンフィギュレーションの例

リスト 3-5 は、5 サイトのドメイン・コンフィギュレーションを定義するコンフィギュレーション・ファイルを示します。Central Bank Branch と通信する、4 つの銀行支店ドメインがあります。銀行支店のうち、3 つがほかの BEA Tuxedo システム・ドメイン内で実行されています。残りの支店は、別の TP ドメインの制御下で実行されており、そのドメインとの通信には OSI-TP が使用されています。この例では、Central Bank から見た BEA Tuxedo のドメイン・ゲートウェイ・コンフィギュレーション・ファイルを示しています。DM_TDOMAIN セクションは、b01 をミラーリングしたゲートウェイを例示しています。

リスト 3-5　5 サイトののドメイン・コンフィギュレーションのための DMCONFIG ファイル

```
# Central Bank 用の BEA Tuxedo ドメイン・コンフィギュレーション・ファイル
#
```

```
#
*DM_LOCAL_DOMAINS
# <local domain name> <Gateway Group name> <domain type> <domain id> <log device>
#                     [<audit log>] [<blocktime>]
#                     [<log name>] [<log offset>] [<log size>]
#                     [<maxrdom>] [<maxrdtran>] [<maxtran>]
#                     [<maxdatalen>] [<security>]
#                     [<tuxconfig>] [<tuxoffset>]

#
#
DEFAULT: SECURITY = NONE
c01     GWGRP = bankg1
        TYPE = TDOMAIN
        DOMAINID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C01"
c02     GWGRP = bankg2
        TYPE = OSITP
        DOMAINID = "BA.CENTRAL01"
        DMTLOGDEV = "/usr/apps/bank/DMTLOG"
        DMTLOGNAME = "DMTLG_C02"
        NWDEVICE = "OSITP"
        URCH = "ABCD"
#
*DM_REMOTE_DOMAINS
#<remote domain name>  <domain type> <domain id>
#
b01     TYPE = TDOMAIN
        DOMAINID = "BA.BANK01"
b02     TYPE = TDOMAIN
        DOMAINID = "BA.BANK02"
b03     TYPE = TDOMAIN
        DOMAINID = "BA.BANK03"
b04     TYPE = OSITP
        DOMAINID = "BA.BANK04"
        URCH = "ABCD"
#
*DM_TDOMAIN
#
#     <local or remote domainname> <network address> [nwdevice]
#
# Local network addresses
c01     NWADDR = "//newyork.acme.com:65432"     NWDEVICE ="/dev/tcp"
c02     NWADDR = "//192.76.7.47:65433"     NWDEVICE ="/dev/tcp"
# Remote network addresses: second b01 specifies a mirrored gateway
b01     NWADDR = "//192.11.109.5:1025" NWDEVICE = "/dev/tcp"
b01     NWADDR = "//194.12.110.5:1025" NWDEVICE = "/dev/tcp"
b02     NWADDR = "//dallas.acme.com:65432"  NWDEVICE = "/dev/tcp"
```

```
b03     NWADDR = "//192.11.109.156:4244" NWDEVICE = "/dev/tcp"
#
*DM_OSITP
#
#<local or remote domain name> <apt> <aeq>
#                              [<aet>] [<acn>] [<apid>] [<aeid>]
#                              [<profile>]
#
c02     APT = "BA.CENTRAL01"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.3},{1}"
        ACN = "XATMI"
b04     APT = "BA.BANK04"
        AEQ = "TUXEDO.R.4.2.1"
        AET = "{1.3.15.0.4},{1}"
        ACN = "XATMI"
*DM_LOCAL_SERVICES
#<service_name>  [<Local Domain name>] [<access control>] [<exported svcname>]
#                [<inbuftype>] [<outbuftype>]
#


open_act        ACL = branch
close_act       ACL = branch
credit
debit
balance
loan            LDOM = c02      ACL = loans
*DM_REMOTE_SERVICES
#<service_name>   [<Remote domain name>] [<local domain name>]
#                 [<remote svcname>] [<routing>] [<conv>]
#                 [<trantime>] [<inbuftype>] [<outbuftype>]
#
tlr_add   LDOM = c01  ROUTING = ACCOUNT
tlr_bal   LDOM = c01  ROUTING = ACCOUNT
tlr_add   RDOM = b04  LDOM = c02 RNAME ="TPSU002"
tlr_bal   RDOM = b04  LDOM = c02 RNAME ="TPSU003"
*DM_ROUTING
# <routing criteria>    <field> <typed buffer> <ranges>
#
ACCOUNT FIELD = branchid  BUFTYPE ="VIEW:account"
        RANGES ="MIN - 1000:b01, 1001-3000:b02, *:b03"
*DM_ACCESS_CONTROL
#<acl name>   <Remote domain list>
#
branch  ACLIST = b01, b02, b03
loans   ACLIST = b04
```

4 CORBA アプリケーションのチューニング

ここでは、次の内容について説明します。

- アプリケーション資源の最大化
- MSSQ セットを使用すべき場合 (BEA Tuxedo ATMI サーバのみ)
- システム制御のロード・バランシングの有効化
- 複製されたサーバ・プロセスおよびグループのコンフィギュレーション
- マルチスレッド・サーバのコンフィギュレーション
- サーバへのサービスのバンドル (BEA Tuxedo ATMI サーバのみ)
- 性能オプション
- アプリケーション・パラメータによる効率の向上
- アプリケーション・パラメータの設定
- IPC 要件の決定
- システム・トラフィックの測定

BEA Tuxedo アプリケーションの監視の詳細については、『BEA Tuxedo アプリケーション実行時の管理』の「BEA TUxedo アプリケーションの監視」を参照してください。

アプリケーション資源の最大化

次の領域で正しい判断を行うことで、BEA Tuxedo アプリケーションの機能が向上します。

- MSSQ セットを使用すべき場合 (BEA Tuxedo ATMI サーバのみ)
- ロード・ファクタの割り当て方法
- インターフェイスやサービスをサーバにパッケージングする方法
- アプリケーション・パラメータの設定方法
- オペレーティング・システムの IPC パラメータのチューニング方法
- ボトルネックの検出および排除方法

MSSQ セットを使用すべき場合 (BEA Tuxedo ATMI サーバのみ)

注記 複数サーバ、単一キュー (MSSQ) セットは、BEA Tuxedo CORBA サーバではサポートされていません。

表 4-1 では、BEA Tuxedo サーバで MSSQ セットを使用する場合について説明します。

表 4-1 MSSQ セットを使用する場合と、使用しない場合

MSSQ セットを使用する場合	MSSQ セットを使用しない場合
複数の、適度な数のサーバがある場合。	サーバ数が多すぎる場合 (ただし、MSSQ セットを多数使用して対処することも可能)。
バッファ・サイズが適度な大きさである場合。	1 つのキューがいっぱいになるほどバッファ・サイズが大きい場合。

表 4-1 MSSQ セットを使用する場合と、使用しない場合 (続き)

MSSQ セットを使用する場合	MSSQ セットを使用しない場合
各サーバが同じサービスのセットを提供する場合。	サーバごとにサービスが異なる場合。
適度なサイズのメッセージが含まれている場合。	キューがいっぱいになるほど長いメッセージがサービスに渡される場合。この場合、非ブロッキング送信が失敗するか、またはブロッキング送信がブロックされます。
サービスのターンアラウンド・タイムが最適であり、一貫性があることが重要視される場合。	サービスのターンアラウンド・タイムが最適であり、一貫性があることが重要視されない場合。

次の 2 つの例で、MSSQ セットの使用が適している場合と適さない場合がある理由を示します。

- MSSQ セットが適切に使用されているアプリケーションは、銀行に似ています。銀行では、すべての窓口で同じサービスが提供されており、顧客は行列に並んで、窓口が空くのを待ちます。この効率的なしくみにより、利用できるサービスを確実に最適な形で使用できます。
- MSSQ セットを使用しないほうがよいアプリケーションは、スーパーマーケットに似ています。スーパーマーケットでは、各レジで、一連の異なるサービスが提供されています。現金のみを受け取るレジもあれば、クレジット・カードを受け付けるレジもあり、さらに 10 個未満の商品を購入する顧客しか扱わないレジもあります。

システム制御のロード・バランシングの有効化

BEA Tuxedo では、システム全体に対し、ロード・バランシングのアルゴリズムを使用するかどうかを制御できます。ロード・バランシングを使用すると、ロード・ファクタがシステム内の各サービスに適用され、各サーバの負荷の合計を監視できます。各サービス要求は、負荷が最も少ない適切なサーバに送信されます。

注記 BEA Tuxedo CORBA システムでは、システム制御のロード・バランシングは自動的に有効化されます。LDBAL=N を指定しても、ロード・バランシングを無効化することはできません。

SERVICES セクションにあるロード・ファクタの割り当て方法を決定するには、アプリケーションを継続的に動作させて、各サービスの実行にかかる平均時間を計算します。算出した平均時間を必要とするサービスの場合は、LOAD 値に 50 (LOAD=50) を割り当てます。算出した平均値よりも長い時間がかかるサービスの場合は、LOAD>50 とします。算出した平均値より短い時間で済むサービスの場合は、LOAD<50 とします。

実行された各サービスにには、ロード・ファクタ (LOAD) が割り当てられます。これにより、各サーバが実行したサービスの負荷の合計がトラッキングされます。各サービス要求は、負荷の合計が最も低いサーバにルーティングされます。ルーティング先のサーバの負荷合計は、要求されたサービスの LOAD ファクタ分だけ増加します。

また、インターフェイスにも LOAD ファクタを適用することができます。LOAD ファクタの詳細については、『BEA Tuxedo アプリケーション実行時の管理』の「コンフィギュレーションの作成」を参照してください。

複製されたサーバ・プロセスおよびグループのコンフィギュレーション

複製されたサーバ・プロセスおよびグループを BEA Tuxedo ドメイン内でコンフィギュレーションするには、次の手順に従います。

1. テキスト・エディタを使用して、アプリケーションの UBBCONFIG ファイルを編集します。
2. GROUPS セクションで、コンフィギュレーションするグループの名前を指定します。
3. SERVERS セクションで、複製するサーバ・プロセスについて、表 4-2 に記載のパラメータを指定します。

表 4-2 SERVERS セクションで指定されるパラメータ

パラメータ	説明
サーバ・アプリケーション名	アプリケーション・サーバを含む実行可能ファイルの名前を指定します。
GROUP	サーバ・プロセスが所属するグループの名前を指定します。複数グループにわたるようにサーバ・プロセスを複製する場合は、各グループにサーバ・プロセスを一度ずつ指定します。
SRVID	数値による識別子を指定し、サーバ・プロセスに一意の ID を付与します。
MIN	アプリケーションの起動時に開始するサーバ・プロセスのインスタンス数を指定します。
MAX	同時に実行できるサーバ・プロセスの最大数を指定します。

MIN パラメータと MAX パラメータは、所定のサーバ・アプリケーションが所定のインターフェイス上で要求を並行処理できるレベルを決定します。実行時には、システム管理者はリソース上のボトルネックを検証し、必要であればさらなるサーバ・プロセスを開始して、アプリケーションをスケーリング

できます。詳細については、『BEA Tuxedo アプリケーション実行時の管理』の「BEA Tuxedo アプリケーションの監視」を参照してください。

注記 MAX パラメータは、インスタンスの最大数を制御します。ただし、BEA Tuxedo はインスタンスを自動的に作成することはありません。システムは、指定された MIN 数値分のインスタンスを自動的に開始します。MIN と MAX の中間の場合は、システム管理者が手動で新規インスタンスを作成する必要があります。いったん MAX に到達すると、tmboot、tmadmin、または TMIB API によってエラーが返されます。

マルチスレッド・サーバのコンフィギュレーション

ここでは、次の内容について説明します。

- データベースの相互運用のための OPENINFO パラメータの設定
- マルチスレッド・サーバのコンフィギュレーションに使用するパラメータ
- インターフェイスへの優先順位の割り当て

マルチスレッド・サーバの詳細については、第 1 章の 13 ページ「マルチスレッド・サーバの使用」を参照してください。

データベースの相互運用のための OPENINFO パラメータの設定

Oracle XA データベース・ソフトウェアと相互運用する場合に、マルチスレッド・サーバによるスレッドの使用を可能にするには、リスト 4-1 に示すように、UBBCONFIG ファイルの GROUPS セクションで OPENINFO パラメータに Threads=true を追加する必要があります。詳細については、Oracle XA のオンライン・マニュアルを参照してください。

リスト 4-1　OPENINFO パラメータへの Threads=true の追加

```
OPENINFO="ORACLE_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=100+LogDir=
.+MaxCur=5+Threads=true"
```

マルチスレッド・サーバのコンフィギュレーションに使用するパラメータ

マルチスレッド CORBA サーバのコンフィギュレーションには、次のパラメータを使用します。これらのパラメータは、UBBCONFIG ファイルで設定します。

- MAXOBJECTS

  注記　MAXOBJECTS パラメータは特にスレッドに適用するものではありませんが、このパラメータを増やしたほうがよい場合があります。マルチスレッド・アプリケーションは、任意の時点において、シングル・スレッド・アプリケーションよりも多くのオブジェクトを活性化する可能性があるためです。

- MAXACCESSERS
- MAXDISPATCHTHREADS
- MINDISPATCHTHREADS
- THREADSTACKSIZE
- CONCURR_STRATEGY

これらのパラメータの設定方法については、次のトピックを参照してください。

- 『BEA Tuxedo アプリケーションの設定』の「コンフィギュレーション・ファイルの作成」および「スレッドを利用する BEA Tuxedo システムの設定方法」
- 『BEA Tuxedo アプリケーション実行時の管理』の「MAXACCESSERS、MAXSERVERS、MAXINTERFACES、および MAXSERVICES パラメータの設定」

インターフェイスへの優先順位の割り当て

ここでは、次の内容について説明します。

- インターフェイスに割り当てる優先順位について
- PRIO パラメータの特性

インターフェイスに割り当てる優先順位について

PRIO パラメータを使用して、BEA Tuxedo インターフェイスに優先順位を割り当てることにより、アプリケーション内のデータ・フローを有効に制御できます。BEA Tuxedo システムで動作する CORBA アプリケーションの場合、PRIO パラメータは、アプリケーションの UBBCONFIG ファイルの INTERFACES セクションで名前を指定した各インターフェイスについて指定できます。

たとえば、サーバ 1 が、インターフェイス A、B、および C を提供するとします。インターフェイス A および B の優先順位は 50、インターフェイス C の優先順位は 70 です。インターフェイス C に対する要求は、A または B に対する要求より常に先にキューから取り出されます。A および B に対する要求は、互いに等しくキューから取り出されます。システムは、10 回に 1 回の割合で先入れ先出し (FIFO) 順序で要求をキューから取り出し、メッセージがキューで無制限に待機しないようにします。

tpsprio() 呼び出しを使用すると、優先順位を動的に変更できます。ただし、選ばれたクライアントだけがインターフェイスの優先順位を高く設定できるようにします。するべきです。サーバがインターフェイス要求を行うシステムでは、サーバ側から tpsprio() を呼び出し、インターフェイス呼び出しの優先順位を上げることができます。これによりユーザは、必要なインターフェイス要求を待たずに済みます。

PRIO パラメータの特性

PRIO パラメータは、慎重に使用してください。キューのメッセージの順序 (たとえば A、B、C) によっては、10 回に 1 回の割合でしか、一部 (A および B など) がキューから取り出されません。つまり、サービスの性能が低下し、ターンアラウンド・タイムが遅くなる可能性があります。

PRIO パラメータの特性は、次のとおりです。

- サーバのキューにおけるインターフェイスの優先順位を決定します。
- 最も高い優先順位が最優先されます。このインターフェイスの発生する頻度は低くなります。
- メッセージは、FIFO に基づき 10 回に 1 回取り出されるため、優先順位の低いメッセージがキューにいつまでもとどまることはありません。応答時間は、優先順位の低いインターフェイスでは問題になりません。

優先順位を割り当てることで、最も重要な要求の処理はより効率的に行い、重要性の低い要求の処理は遅らせて行うことができます。また、特定のユーザまたは特定の状況に対して優先順位を付けることも可能です。

サーバへのサービスのバンドル (BEA Tuxedo ATMI サーバのみ)

ここでは、次の内容について説明します。

- サービスのバンドルについて
- サービスのバンドルが必要な場合

サービスのバンドルについて

複数のサービスをサーバの実行可能ファイルにパッケージ化する最も簡単な方法は、まったくパッケージ化しないことです。しかし、サービスをパッケージ化しないと、サーバの実行可能ファイル、メッセージ・キューおよびセマフォの数が許容範囲を超える原因となります。サービスをまったくバンドルしない (まとめない) 場合と細かくバンドルする (まとめる) 場合では、それぞれ長所と短所があります。

サービスのバンドルが必要な場合

サービスをバンドルする理由は、以下のとおりです。

- 機能が類似している－ アプリケーション内に役割の類似するサービスがある場合は、それらのサービスを同じサーバにバンドルできます。アプリケーション側は、指定された時点で、すべてのサービスを提供するか、またはサービスをまったく提供しないかのどちらかを行います。たとえば bankapp アプリケーションでは、WITHDRAW、DEPOSIT、および INQUIRY の各サービスはすべて、窓口オペレーションです。サービスの管理は、簡略化されます。
- ライブラリが類似している － たとえば、同じ 100K のライブラリを使用する 3 つのサービスと、異なる 100K のライブラリを使用する 3 つのサービスがある場合、最初の 3 つのサービスをバンドルすると 200K の領域を節約できます。たいていの場合、同等の機能を持つサービスには、類似するライブラリがあります。
- キューー キューで処理できる範囲のサービスのみをサーバにバンドルしてください。空の MSSQ セットに追加された各サービスは、サーバの実行可能モジュールのサイズにはあまり影響しません。また、システム上のキューの数にはまったく影響しません。ただし、キューがいっぱいになるとシステムの性能は低下するため、対応策として別の実行可能モジュールを作成する必要があります。
- 呼び出し依存型サービスの設定－ 同じサーバに、お互いを呼び出すサービスを 2 つ以上設定しないでください。設定すると、サーバは自身を呼び出し、デッドロックの原因となります。

性能オプション

BEA Tuxedo のリリース 8.0 から、性能オプションが追加されました。これらのオプションにより、BEA Tuxedo のインフラストラクチャで特定の機能を無効化できます。無効化するのは、CORBA または ATMI アプリケーションで必要でない機能のみとしてください。表 4-3 で、これらのオプションを説明します。

表 4-3 性能オプション

オプション	説明	設定方法
サービスとインターフェイスのキャッシングのオプション (SICACHEENTRIESMAX および TMSICACHEENTRIESMAX)	このオプションにより、サービスおよびインターフェイスのエントリをキャッシングし、掲示板をロックすることなくサービスまたはインターフェイスのキャッシングしたコピーを使用できます。	これらのオプションの詳細については、『BEA Tuxedo アプリケーション実行時の管理』および『BEA Tuxedo のファイル形式とデータ記述方法』の「UBBCONFIG(5)」、「TM_MIB(5)」、および「tuxenv(5)」を参照してください。
スレッドの無効化 (TMNOTHREADS)	マルチスレッド処理を無効化するには、このオプションを「yes」に設定します。このスレッドを使用しないアプリケーションで、これらを無効にすると、性能が著しく向上します。	このオプションを設定するには、tuxenv(5) を使用します。詳細については、『BEA Tuxedo アプリケーション実行時の管理』および『BEA Tuxedo のファイル形式とデータ記述方法』の「tuxenv(5)」を参照してください。
認可と監査の無効化 (オプション {[NO_AA]})	このオプションを設定すると、アプリケーションごとに監査および認可の機能を無効化できます。	このオプションの設定は、UBBCONFIG ファイルの RESOURCES セクションで行います。詳細については、『BEA Tuxedo アプリケーション実行時の管理』および『BEA Tuxedo のファイル形式とデータ記述方法』に記載の UBBCONFIG(5) の RESOURCES セクション内の OPTION を参照してください。

表 4-3 性能オプション (続き)

オプション	説明	設定方法
XA トランザクションの無効化 (NO_XA)	このオプションを設定すると、XA トランザクションを無効化できます。	NO_XA オプションの詳細については、『BEA Tuxedo アプリケーション実行時の管理』および『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5) と TM_MIB(5) を参照してください。

アプリケーション・パラメータによる効率の向上

ここでは、次の内容について説明します。

- IPC 要件の決定
- MINDISPATCHTHREADS
- MAXDISPATCHTHREADS
- MAXGTT、MAXBUFTYPE、および MAXBUFSTYPE パラメータの設定
- SANITYSCAN、BLOCKTIME、BBLQUERY、および DBBLWAIT パラメータの設定

これらのアプリケーション・パラメータを設定することにより、システム効率を向上できます。

MAXDISPATCHTHREADS

MAXDISPATCHTHREADS パラメータは、各サーバ・プロセスで生成される、同時に実行できるディスパッチ・スレッドの最大数を決定します。このパラメータを指定する際には、次の事項を考慮してください。

- MAXDISPATCHTHREADS の値により、スレッド・プールが受信する要求に対応して拡張されていく上での、許容される最大サイズが決まります。

- MAXDISPATCHTHREADS のデフォルト値は 1 です。1 より大きい値を指定すると、システムは特殊なディスパッチャ・スレッドを作成して使用します。このディスパッチャ・スレッドは、スレッド・プールの最大サイズを決定するスレッド数には、含まれていません。
- MAXDISPATCHTHREADS パラメータの値に 1 を指定すると、サーバ・アプリケーションがシングル・スレッド・サーバとしてコンフィギュレーションされることを示します。1 より大きい値を指定すると、サーバ・アプリケーションがマルチスレッド・サーバとしてコンフィギュレーションされることを示します。
- MAXDISPATCHTHREADS パラメータに指定する値は、MINDISPATCHTHREADS パラメータに指定する値を下回っていてはいけません。
- オペレーティング・システムにより、1 つのプロセスで作成できるスレッドの最大数は制限されます。MAXDISPATCHTHREADS は、その制限値より少ない、アプリケーションが必要とするアプリケーション管理スレッドの数を差し引いた値にします。

MAXDISPATCHTHREADS パラメータの値は、ほかのパラメータにも影響を与えます。たとえば、MAXACCESSORS パラメータは BEA Tuxedo システムへの同時アクセス数を制御します、各スレッドは、1 つのアクセサとしてカウントされます。マルチスレッド・サーバのアプリケーションの場合、各サーバの実行がコンフィギュレーションされているシステム管理のスレッドの数を考慮しておく必要があります。システム管理のスレッドとは、アプリケーションで開始され管理されるスレッドに対して、BEA Tuxedo ソフトウェアで開始され管理されるスレッドです。内部では BEA Tuxedo が、利用可能なシステム管理のスレッドのプールを管理しています。クライアント要求が受信されると、スレッド・プールから利用可能なシステム管理のスレッドがその要求を実行するように、スケジューリングされています。要求が完了すると、システム管理のスレッドは利用可能なスレッドのプールに返されます。

たとえば、システム内に 4 つのマルチスレッド・サーバがあり、各サーバが 50 個のシステム管理のスレッドを実行するようにコンフィギュレーションされている場合、これらのサーバにおけるアクセサ要件は、次のように計算される、アクセサ数の合計となります。

```
50 + 50 + 50 + 50 = 200 アクセサ
```

MINDISPATCHTHREADS

サーバが最初にブートされたときに開始されるサーバ・ディスパッチ・スレッドの数を指定するには、MINDISPATCHTHREADS パラメータを使用します。このパラメータを指定する際には、次の事項を考慮してください。

- MINDISPATCHTHREADS の値で、スレッド・プール内のスレッドの最初の割り当てが決まります。
- MAXDISPATCHTHREADS が 1 より大きい場合に作成される別個のディスパッチャ・スレッドは、MINDISPATCHTHREADS の範囲には含まれません。
- MINDISPATCHTHREADS に指定する値は、MAXDISPATCHTHREADS に指定する値を上回ってはいけません。
- MINDISPATCHTHREADS のデフォルト値は 0 です。

MAXACCESSERS、MAXOBJECTS、MAXSERVERS、MAXINTERFACES、および MAXSERVICES パラメータの設定

MAXACCESSERS、MAXOBJECTS、MAXSERVERS、MAXINTERFACES、および MAXSERVICES の各パラメータを使用すると、セマフォおよび共用メモリのコストが増大するため、システムの要求を満たす最低限の値を選択してください。また、システムに同時にアクセスできるクライアント数を設定できるようにしておく必要もあります。これらのパラメータには、デフォルトで IPC 資源が適度に割り当てられています。しかし、アプリケーションにとって必要最低限の値を設定するのが賢明です。

マルチスレッド・サーバの場合、各サーバの実行がコンフィギュレーションされているスレッドの数を考慮しておく必要があります。MAXACCESSERS パラメータは、BEA Tuxedo システムの同時アクセサの最大数を設定します。アクセサには、ネイティブおよびリモートのクライアント、サーバ、および管理プロセスが含まれます。MAXACCESSERS パラメータの設定の詳細については、第 4 章の 12 ページ「MAXDISPATCHTHREADS」を参照してください。

MAXGTT、MAXBUFTYPE、および MAXBUFSTYPE パラメータの設定

システム内のクライアント数に、それらのクライアントがトランザクションをコミットする回数のパーセンテージを乗算した積が 100 に近似している場合には、MAXGTT パラメータの値を大きくする必要があります。MAXGTT の値を大きくした場合には、それに応じて、各マシンの TLOGSIZE の値も大きくする必要があります。分散トランザクションを使用しないアプリケーションの場合は、MAXGTT を 0 に設定します。

アプリケーションで受け付けられるバッファのタイプおよびサブタイプの数はそれぞれ、MAXBUFTYPE パラメータおよび MAXBUFSTYPE パラメータで制限できます。MAXBUFTYPE の現在のデフォルト値は 16 です。ユーザ定義のバッファ・タイプが多数指定されていない限り、MAXBUFTYPE は省略できます。しかし、何種類もの VIEW サブタイプを使用する予定がある場合は、MAXBUFSTYPE の設定を、現在のデフォルト値である 32 より増やしておきます。

SANITYSCAN、BLOCKTIME、BBLQUERY、および DBBLWAIT パラメータの設定

システムが、使用率が高いなどの理由で処理の遅いプロセッサで稼動している場合は、タイミング・パラメータである SANITYCAN、BLOCKTIME、およびトランザクションのタイムアウト値を増やします。ネットワークの動作が遅い場合は、BLOCKTIME、BBLQUERY および DBBLWAIT パラメータの値を増やします。

アプリケーション・パラメータの設定

表 4-4 では、アプリケーションのチューニングに使用できるシステム・パラメータを説明します。

表 4-4 アプリケーションのチューニングに使用するシステム・パラメータ

パラメータ	操作
MAXACCESSERS、MAXOBJECTS、MAXSERVERS、MAXINTERFACES、および MAXSERVICES	必要最低限の値を設定します (IPC 資源を確保するため)。 クライアントを追加できるように設定します。
MAXGTT、MAXBUFTYPE、および MAXBUFSTYPE	MAXGTT の値を増やして、多数のクライアントを許容します。トランザクション処理を行わないアプリケーションでは、MAXGTT を 0 に設定します。 ユーザ定義のバッファ型を 8 つ以上作成する場合に限り、MAXBUFTYPE を使用します。 何種類もの VIEW サブタイプを使用する場合は、MAXBUFSTYPE の値を増やします。
BLOCKTIME、TRANTIME、および SANITYSCAN	システムの動作が遅い場合は値を増やします。
BLOCKTIME、TRANTIME、BBLQUERY、および DBBLWAIT	ネットワークの動作が遅い場合は値を増やします。

IPC 要件の決定

IPC 要件は、さまざまなシステム・パラメータの値で決まります。tmboot -c コマンドを使用すると、コンフィギュレーションの IPC 要求をテストできます。次のパラメータの値は、アプリケーションで必要とされる IPC に影響をもたらします。

- MAXACCESSERS
- REPLYQ
- RQADDR (これを使用すると MSSQ セットを形成できます)
- MAXSERVERS
- MAXSERVICES
- MAXGTT

表 4-5 では、アプリケーションで必要とされる IPC に影響を与えるシステム・パラメータについて説明します。

表 4-5 IPC パラメータのチューニング

パラメータ	アクション
MAXACCESSERS	セマフォの数を指定します。 メッセージ・キューの数は、「MAXACCESSERS + 応答キューを持つサーバ数 (MSSQ セットのサーバ数 + MSSQ セット数)」とほぼ同じです。
MAXSERVERS、MAXSERVICES、および MAXGTT	MAXSERVERS、MAXSERVICES、MAXGTT と、ROUTING、GROUP、および NETWORK セクションの全体のサイズは共用メモリのサイズに影響しますが、これらのパラメータの相関関係を計算式に表わそうとすると複雑になります。そこで代わりに、単に tmboot -c または tmloadcf -c を実行して、アプリケーションの最低 IPC 要件を計算します。

表 4-5 IPC パラメータのチューニング (続き)

パラメータ	アクション
キューに関連するカーネル・パラメータ	クライアントとサーバ間のバッファ・トラフィックのフローを管理するためにチューニングします。キューの最大サイズ (バイト単位) は、アプリケーションで最も大きいメッセージを処理でき、通常は 75 ～ 85 パーセントが使用中になる大きさに設定します。それよりパーセンテージを低くすると、無駄が生じます。それより大きくすると、ブロックへのメッセージ送信頻度が高くなりすぎます。 アプリケーションが送信するメッセージに対して最大のバッファを処理できるように最大サイズを設定します。 キューの最大長 (キューに同時にとどまることができる最大メッセージ数) は、アプリケーションにおけるオペレーションに適した大きさに設定します。 キューの平均使用率と平均長を測定するには、アプリケーションをシミュレートするか、または実行します。これは、アプリケーションの実行前にチューニング可能なパラメータを予測し、アプリケーションの実行後に性能分析に基づいてそのパラメータを調整するという、試行錯誤のプロセスになります。 大規模なシステムでは、オペレーティング・システム・カーネルのサイズに対するパラメータ設定の効果を分析します。効果が認められない場合には、アプリケーション・プロセスの数を減らすか、またはアプリケーションをさらに多くのマシンに分散して MAXACCESSERS を減らします。

システム・トラフィックの測定

ここでは、次の内容について説明します。

- システム・トラフィックとボトルネックについて
- システム・ボトルネックの検出例
- UNIX におけるボトルネックの検出
- Windows におけるボトルネックの検出

BEA Tuxedo アプリケーションの監視とトラフィック測定の詳細については、『BEA Tuxedo アプリケーション実行時の管理』の「BEA Tuxedo アプリケーションの監視」を参照してください。

システム・トラフィックとボトルネックについて

トラフィックの量が資源容量の上限に近くなると、システムにボトルネックが生じる可能性があります。サービス・トラフィックを測定するには、インプリメンテーション・コードでグローバル・カウンタを使用します。

たとえば Tuxedo アプリケーションでは、ブート時に `tpsvrinit()` が呼び出されると、グローバル・カウンタを初期化して、開始時間を記録できます。以降、特定のサービスが呼び出されるたびにカウンタ値は増加します。`tpsvrdone()` 関数を呼び出してサーバをシャットダウンすると、最終カウンタ値と終了時間が記録されます。このメカニズムにより、一定期間に特定サービスのビジー状態がどの程度であるかを判断できます。

注記 CORBA C++ アプリケーションの場合は、`Server::initialize()` オペレーションと `Server::release()` オペレーションを使用します。

BEA Tuxedo では、データ・フローのパターンが原因でボトルネックが生じます。ボトルネックを検出する最も早い方法は、クライアント側から関連するサービスに要する時間を測定することです。

システム・ボトルネックの検出例

クライアント 1 では、画面の出力に 4 秒必要であるとします。`time`(2) を呼び出すと、サービス A に対する `tpcall` が動作を 3.7 秒遅らせていることがわかります。サービス A を開始点と終了点で監視すると、0.5 秒かかっています。これは、キューがいっぱいであることを示唆しています。この判断は、`pq` コマンドを使用して行われます。

一方、サービス A の実行に 3.2 秒かかるとします。サービス A の個々の部分はまとめて測定できます。サービス A がサービス B に対して `tpcall` を発行し、この動作に 2.8 秒かかっていることも考えられます。この場合、キュー時間またはメッセージ送信ブロッキング時間を分離できます。適切な時間を識別すると、アプリケーションはトラフィックを処理できるように再度チューニングされます。

time(2) を使用すると、次の項目の所要時間を測定できます。

- クライアント・プログラム全体
- クライアント・サービス要求のみ
- サービス機能全体
- サービス要求を行うサービス機能 (ある場合)

UNIX におけるボトルネックの検出

UNIX システムの sar(1) コマンドを使用すると、システムのボトルネックの検出に役立つ性能についての情報が表示されます。sar(1) は、次の目的に使用できます。

- あらかじめ決められた間隔でオペレーティング・システムの累積アクティビティ・カウンタをサンプリングする。
- システム・ファイルからデータを抽出する。

表 4-6 では、sar(1) コマンドのオプションを説明します。

表 4-6 sar(1) コマンドのオプション

オプション	説明
-u	CPU の使用率を示します。ユーザ・モード、システム・モード、待機状態 (ブロック I/O を待つプロセスを持ったままアイドル状態)、およびアイドル状態である時間を割合で示します。
-b	バッファのアクティビティ (システム・バッファとディスクまたはほかのブロック・デバイスとの間で送信される 1 秒あたりのデータ転送数など) を報告します。
-c	システム・コールのアクティビティを報告します。これには、すべての種類のシステム・コールと、fork(2) や exec(2) などの特定のシステム・コールが含まれます。

表 4-6 sar(1) コマンドのオプション (続き)

オプション	説明
-w	システムのスワッピング・アクティビティを監視します。これは、スワップ・インとスワップ・アウトの転送数などです。
-q	専有時のキューの長さの平均および専有時間の割合を報告します。
-m	メッセージとシステム・セマフォのアクティビティ (1 秒あたりのプリミティブの数) を報告します。
-p	ページング・アクティビティ (アドレス変換ページ障害、ページ障害と保護エラー、およびフリー・リストに再利用された有効ページなど) を報告します。
-r	未使用のメモリ・ページとディスク・ブロック (ユーザ・プロセスで使用できる平均ページ数、プロセス・スワッピングに対して使用できるディスク・ブロックなど) を報告します。

注記 UNIX プラットフォームの中には、sar(1) コマンドの代わりに別のコマンドが用意されているものもあります。たとえば、BSD には iostat(1) コマンド、Sun Microsystems, Inc. には perfmeter(1) が用意されています。

Windows におけるボトルネックの検出

Windows でシステム情報を収集しボトルネックを検出するには、パフォーマンス・モニタを使用します。[スタート] メニューの [設定] をクリックして [コントロール パネル] をクリックします。次に [管理ツール] の [パフォーマンス] をクリックします。

索引

A

AUTOTRAN パラメータ 3-11, 3-12

B

BBLQUERY パラメータ 4-15, 4-16
BLOCKTIME パラメータ 4-15, 4-16

C

CLOSEINFO パラメータ 3-10
create_object_reference() オペレーション 2-15

D

DBBLWAIT パラメータ 4-15, 4-16
DMCONFIG ファイル
  DM_ROUTING セクション 3-16
  DMCONFIG ファイルについて 3-15
  例 3-19

F

factory_finder.ini 3-14

G

GROUP パラメータ 4-5
GRPNO パラメータ 3-9

I

IIOP ハンドラ (ISH)
  ISH について 1-24
  ISH プロセス数の増加 1-25
IIOP リスナ (ISL) 1-24
iostat(1) コマンド 4-21
IPC 要件
  キューに関連するカーネル・パラメータのチューニング 4-18
  決定 4-17–4-18
  パラメータのチューニング 4-17

L

LMID パラメータ 3-9

M

MAX パラメータ 4-5
MAXACCESSERS パラメータ 4-14, 4-16, 4-17
MAXACCESSORS パラメータ 4-13
MAXBUFSTYPE パラメータ 4-15, 4-16
MAXBUFTYPE パラメータ 4-15, 4-16
MAXDISPATCHTHREADS パラメータ 4-12
  ほかのパラメータへの影響 4-13
MAXGTT パラメータ 4-15, 4-16, 4-17
MAXINTERFACES パラメータ 4-14, 4-16
MAXSERVERS パラメータ 4-14, 4-16, 4-17
MAXSERVICES パラメータ 4-14, 4-16, 4-17
MIN パラメータ 4-5
MINDISPATCHTHREADS パラメータ 4-14
MSSQ セット
  使い方 4-2
  例 4-3

O

OMG IDL、Production サンプル・アプリケーション 2-4
OPENINFO パラメータ 3-10

P

perfmeter(1) コマンド 4-21
PRIO パラメータ 4-8
Production サンプル・アプリケーション
  OMG IDL の変更 2-4
  UBBCONFIG ファイル 2-9
  アプリケーションをさらにスケーリングする方法 2-23
  サーバ・グループの複製 2-8
  サーバ・プロセスの複製 2-6
  状態を持たないオブジェクト・モデル 2-4
  スケーリングの方法 2-2
  設計上の追加考慮事項 2-18
  設計上の目標 2-2
  ファクトリ・ベース・ルーティング 2-11

S

SANITYSCAN パラメータ 4-15, 4-16
sar(1) コマンド 4-20
SRVID パラメータ 4-5

T

tmboot(1) -c コマンド 4-17
TMSCOUNT パラメータ 3-9
TMSNAME パラメータ 3-9
TRANTIME パラメータ 3-11, 3-12, 4-16
tsprio 呼び出し 4-8

U

UBBCONFIG ファイル
  GROUPS セクション
    CLOSEINFO パラメータ 3-10
    GRPNO パラメータ 3-9
    LMID パラメータ 3-9
    OPENINFO パラメータ 3-10, 4-6
    TMSCOUNT パラメータ 3-9
    TMSNAME パラメータ 3-9
  Production サンプル・アプリケーション 2-9
  ROUTING セクション 3-13
  SERVERS セクション
    GROUP パラメータ 4-5
    MAX パラメータ 4-5
    MIN パラメータ 4-5
    SRVID パラメータ 4-5
  SERVICES セクション
    AUTOTRAN パラメータ 3-11, 3-12
    TRANTIME パラメータ 3-11, 3-12
    例 3-11, 3-13
  分散アプリケーションの例 3-6

あ

アクセサ
  要件の計算 4-13
アプリケーションのスケーラビリティ要件 1-2
アプリケーションのチューニング 4-1–4-21
  IPC 要件の決定 4-17
  アプリケーション資源の最大化 4-2
    サーバへのサービスのバンドル 4-9
    ロード・バランシングの有効化 4-4
  アプリケーション・パラメータの使用 4-12, 4-14, 4-15
  システム・トラフィックの測定 4-18
    システム・ボトルネックの検出 4-19
アプリケーションの分散
  UBBCONFIG ファイル 3-6
  アプリケーションの分散について 3-2
  アプリケーション分散の特性 3-3

サンプル・アプリケーション 3-5
ドメイン・ゲートウェイ・ファイルとルーティング 3-15
複数ドメインでのファクトリ・ベース・ルーティング 3-14
利点 3-2
アプリケーション・パラメータ
設定 4-16
使い方 4-12

い

インターフェイス、優先順位の割り当て 4-8

お

オブジェクト
状態を持たないオブジェクト 1-5
状態を持つオブジェクト 1-5
トランザクション・バウンド 1-5
プロセス・バウンド 1-4
メソッド・バウンド 1-4
オブジェクト状態管理
Production サンプル・アプリケーション 2-4
オブジェクト状態モデル
状態を持たないオブジェクト 1-5
状態を持つオブジェクト 1-5

か

カーネル・パラメータ、チューニング 4-18
カスタマ・サポートへのお問い合わせ情報 ix
管理パラメータ
MAXACCESSORS 4-13
MAXDISPATCHTHREADS 4-12
MINDISPATCHTHREADS 4-14
関連情報 viii

こ

コンフィギュレーション・ファイルのパラメータ
FACTORYROUTING 3-12
MAXOBJECTS 4-16

さ

サーバ・グループ
複製 1-12
複製について 1-9
サーバ・プロセス
複製 1-11
複製について 1-9
サービスのバンドル
サービスのバンドルが必要な場合 4-10
サービスのバンドルについて 4-9
サポート
テクニカル ix

し

資源、アプリケーションの最大化 4-2–4-16
受信するクライアント接続の多重化 1-24
状態を持たないオブジェクト
使用すべき場合 1-6
状態を持たないオブジェクトについて 1-5
状態を持つオブジェクト
使用すべき場合 1-7
状態を持つオブジェクトについて 1-5

す

スケーラビリティ
特長 1-3
要件 1-2

せ

製品マニュアルを印刷する viii

て

データ依存型ルーティング
サンプル・アプリケーション 3-5
使用 (Tuxedo のみ) 3-4
データ依存型ルーティングについて 3-4
特性 3-5

と

ドメイン・ゲートウェイ・コンフィギュレーション・ファイル (DMCONFIG) 3-15
トラフィック、システムの測定 4-18–4-21
トランザクション・バウンド・オブジェクト 1-5

ふ

ファクトリ・ベース・ルーティング
Production サンプル・アプリケーション 2-11
コンフィギュレーション 1-19
Production サンプル・アプリケーション 2-12
しくみ 1-18
特性 1-17
ファクトリでのインプリメンテーション 2-15
ファクトリ・ベース・ルーティングについて 1-16
複数ドメイン用のコンフィギュレーション 3-14
複数サーバ、単一キュー (MSSQ) 4-2
複製
コンフィギュレーションのオプション 1-10
サーバ・グループ
Production サンプル・アプリケーション 2-8
サーバ・グループの複製について 1-12
サーバ・プロセス
Production サンプル・アプリケーション 2-6
ガイドライン 1-12
サーバ・プロセスの複製について 1-11
利点 1-11
サーバ・プロセスとサーバ・グループの複製について 1-9
プロセス・バウンド・オブジェクト 1-4

ほ

ボトルネック、検出 4-19

ま

マニュアルの場所 viii
マルチスレッド処理
コーディングの推奨事項 1-15
コンフィギュレーション
OPENINFO パラメータ 4-6
使用すべき場合 1-14
マルチスレッド Java サーバについて 1-13

め

メソッド・バウンド・オブジェクト 1-4

ゆ

優先順位
PRIO パラメータ 4-8
インターフェイスまたはサービスへの割り当て 4-8

ろ

ロード・バランシング
有効化 4-4