Sun Java System Application Server Enterprise Edition 8.2 配備計画ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–1268

070828@18135

目次

図目次

表目次

例目次

はじめに

『配備計画ガイド』では、本番配備を構築する方法について説明します。

この「はじめに」では、Sun Java™ System Application Server マニュアルセット全体についての情報および表記規則を示します。

Application Server のマニュアルセット

Application Server のマニュアルセットは、配備の計画とシステムのインストールについて説明しています。スタンドアロンの Application Server マニュアルの URL は `http://docs.sun.com/app/docs/coll/1310.4` です。Sun Java Enterprise System (Java ES) Application Server マニュアルの URL は `http://docs.sun.com/app/docs/coll/1659.1` です。Application Server の概要については、次の表に示されている各マニュアルを上から順に参照してください。

表 P–1　Application Server のマニュアルセットの内容

マニュアル名	説明
『リリースノート』	ソフトウェアとマニュアルに関する最新情報。サポート対象のハードウェア、オペレーティングシステム、Java Development Kit (JDK™)、およびデータベースドライバについての総合的な概要情報を表形式で示しています。
『Quick Start Guide』	Application Server 製品の使用を開始するための手順。
『Installation Guide』	ソフトウェアとそのコンポーネントのインストール。
『配備計画ガイド』	最適な方法で確実に Application Server を導入するための、システムニーズや企業ニーズの分析。サーバーを配備する際に注意すべき一般的な問題や懸案事項についても説明しています。
『Developer's Guide』	J2EE コンポーネントおよび API 用のオープン Java 標準モデルに従い、Application Server 上で実行することを目的とする Java 2 Platform, Enterprise Edition (J2EE™ プラットフォーム) アプリケーションの作成と実装。開発ツール、セキュリティー、デバッグ、配備、ライフサイクルモジュールの作成などについての情報も提供します。

表 P–1 Application Server のマニュアルセットの内容 (続き)

マニュアル名	説明
『J2EE 1.4 Tutorial』	J2EE アプリケーションの開発のための J2EE 1.4 プラットフォーム技術および API の使用。
『管理ガイド』	管理コンソールからの、Application Server サブシステムおよびコンポーネントの設定、管理、および配備。
『高可用性 (HA) 管理ガイド』	高可用性データベースのための、インストール後の設定と管理に関する解説。
『Administration Reference』	Application Server 設定ファイル `domain.xml` の編集。
『アップグレードと移行』	新しい Application Server プログラミングモデルへのアプリケーションの移行 (特に Application Server 6.x または 7 からの移行)。このマニュアルでは、製品仕様の非互換性をもたらす可能性のある、隣接した製品リリース間の相違点や設定オプションについても説明しています。
『パフォーマンスチューニングガイド』	パフォーマンスを向上させるための Application Server の調整。
『トラブルシューティングガイド』	Application Server の問題の解決。
『Error Message Reference』	Application Server のエラーメッセージの解決。
『Reference Manual』	Application Server で使用できるユーティリティーコマンド。マニュアルページのスタイルで記述されています。`asadmin` コマンド行インタフェースも含みます。

関連マニュアル

Application Server は、単体で購入することが可能です。あるいは、ネットワークまたはインターネット環境にわたって分散しているエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャーである Java ES のコンポーネントとして購入することもできます。Application Server を Java ES のコンポーネントとして購入した場合は、`http://docs.sun.com/coll/1657.1` にあるシステムマニュアルをよく読むことをお勧めします。Java ES およびそのコンポーネントに関するすべてのマニュアルの URL は `http://docs.sun.com/prod/entsys.5` です。

その他の Sun Java System サーバーのマニュアルについては、次を参照してください。

- Message Queue のマニュアル
- Directory Server のマニュアル
- Web Server のマニュアル

さらに、次のリソースが役立つことがあります。

- J2EE 1.4 各種仕様 (`http://java.sun.com/j2ee/1.4/docs/index.html`)
- J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`)
- J2EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

デフォルトのパスとファイル名

次の表は、このマニュアルで使用するデフォルトのパスやファイル名について説明したものです。

表 P–2 デフォルトのパスとファイル名

プレースホルダ	説明	デフォルト値

install-dir	Application Server のベースインストールディレクトリを表します。	Solaris™ プラットフォームへの Sun Java Enterprise System (Java ES) インストールの場合:
		`/opt/SUNWappserver/appserver`
		Linux プラットフォームへの Java ES インストールの場合:
		`/opt/sun/appserver/`
		Solaris および Linux プラットフォームへのインストールで、ルートユーザーでない場合:
		ユーザーのホームディレクトリ `/SUNWappserver`
		Solaris および Linux プラットフォームへのインストールで、ルートユーザーである場合:
		`/opt/SUNWappserver`
		Windows のすべてのインストールの場合:
		SystemDrive`:\Sun\AppServer`

domain-root-dir	すべてのドメインを含むディレクトリを表します。	Solaris プラットフォームへの Java ES インストールの場合:
		`/var/opt/SUNWappserver/domains/`
		Linux プラットフォームへの Java ES インストールの場合:
		`/var/opt/sun/appserver/domains/`
		そのほかのすべてのインストールの場合:
		install-dir`/domains/`

domain-dir	ドメインのディレクトリを表します。 設定ファイルには、次のように表される *domain-dir* があります。 `${com.sun.aas.instanceRoot}`	*domain-root-dir*`/`*domain-dir*
instance-dir	サーバーインスタンスのディレクトリを表します。	*domain-dir*`/`*instance-dir*

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–3 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% you have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	machine_name% **su** Password:
aabbcc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャー・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

記号の規則

次の表は、この用語集で使用される記号の一覧です。

表 P–4 記号の規則

記号	説明	例	意味
[ ]	省略可能な引数やコマンドオプションが含まれます。	ls [-l]	-l オプションは必須ではありません。
{ | }	必須のコマンドオプションの選択肢を囲みます。	-d {y|n}	-d オプションには y 引数か n 引数のいずれかを使用する必要があります。
${ }	変数参照を示します。	${com.sun.javaRoot}	com.sun.javaRoot 変数の値を参照します。
-	同時に押すキーを示します。	Control-A	Control キーを押しながら A キーを押します。
+	順番に押すキーを示します。	Ctrl+A+N	Control キーを押してから放し、それに続くキーを押します。
→	グラフィカルユーザーインタフェースでのメニュー項目の選択順序を示します。	「ファイル」→「新規」→「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

マニュアル、サポート、およびトレーニング

Sunのサービス	URL	内容
マニュアル	http://jp.sun.com/documentation/	PDF文書およびHTML文書をダウンロードできます。
サポートおよびトレーニング	http://jp.sun.com/supportraining/	技術サポート、パッチのダウンロード、およびSunのトレーニングコース情報を提供します。

◆ ◆ ◆ 第 1 章

1 製品概念

Sun Java System Application Server は、J2EE アプリケーションの開発、配備、および管理のための堅牢なプラットフォームを提供します。主な機能には、スケーラブルなトランザクション管理、コンテナ管理による持続性ランタイム、Web サービスのパフォーマンス、クラスタ化、高可用性セッション状態、セキュリティー、および統合機能があります。

Application Server は 3 種類のエディションで提供されており、各エディションは、特定の使用状況およびサービスレベルに固有の機能性を提供するように設計されています。

Platform Edition は無償で配布され、ソフトウェア開発および部門レベルの本稼動環境を構築するために使用できます。このエディションは、Java EE アプリケーションを配備するための本番プラットフォームを提供し、Solaris オペレーティングシステムと統合されます。開発者が扱いやすい軽量 J2EE コンテナが含まれています。プロセス数が限定されている分設定が簡易であり、迅速で効率的な配備を可能にするインタフェースを備えています。Application Server Platform Edition は、NetBeans 5.5 統合開発環境 (IDE) を含む一部の開発ツールとも統合されています。

Standard Edition と Platform Edition の違いは次のとおりです。

- Application Server Standard Edition は、Application Server Platform Edition よりもスケーラビリティーに優れた高性能 Web コンテナを提供します。
- Application Server Standard Edition では複数マシンの管理機能が強化されており、リモートサーバーの 1 つのコンソールから複数台のサーバーを制御できます。

Application Server Standard Edition は、サービスの可用性を高める次の機能を提供します。

- 負荷分散機能により、クライアント要求を複数のアプリケーションサーバーに転送し、最小の応答時間とスループットの向上を達成できます。
- 複数のアプリケーションサーバーをクラスタ化して実行し、スケーラビリティーとフェイルオーバーを向上させることができます。

Application Server Enterprise Edition は、データ可用性に関する機能を Application Server Standard Edition に追加したものです。Application Server Enterprise Edition では、高可用性 (HA) セッションストアとして高可用性データベース (HADB) を使用します。HTTP セッションおよび SFSB の対話状態が HADB に格納され、あとから回復可能です。可用性により、クラスタ内のアプリケーションサーバーインスタンスのフェイルオーバー保護が提供されます。Application Server Enterprise Edition は、セッション状態の保護が必要とされるエンタープライズ規模のアプリケーションおよびサービス配備にもっとも適しています。

この章の内容は次のとおりです。

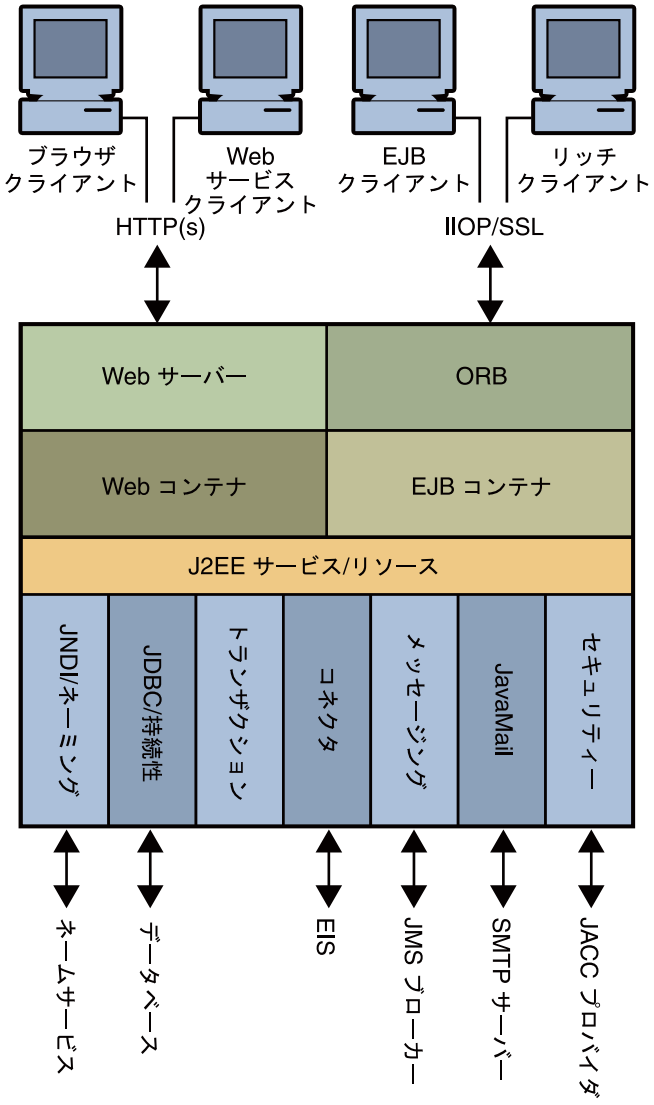
J2EE プラットフォームの概要

Application Server は、多層エンタープライズアプリケーションを開発するための標準規格を定義する Java 2 Enterprise Edition (J2EE) 1.4 テクノロジを実装します。J2EE プラットフォームは、標準化されたモジュール型コンポーネントに基づく開発を可能にし、それらのコンポーネントに対するサービスの完全なセットを提供し、複雑なプログラミング不要でアプリケーション動作の細部の多くを自動的に処理することによって、エンタープライズアプリケーションを簡素化します。

J2EE アプリケーション

J2EE アプリケーションは、JavaServer Pages (JSP)、Java サーブレット、Enterprise JavaBeans (EJB) モジュールなどのコンポーネントで構成されます。ソフトウェア開発者はこれらのコンポーネントを利用して、大規模分散アプリケーションを構築できます。開発者は J2EE アプリケーションを、zip ファイルに似た Java アーカイブ (JAR) ファイルにパッケージ化して本番サイトに配布できます。管理者は、J2EE JAR ファイルを 1 つ以上のサーバーインスタンス (またはインスタンスのクラスタ) に配備することにより、J2EE アプリケーションを Application Server 上にインストールします。

次の図は、以降の節で説明する J2EE プラットフォームのコンポーネントを示したものです。



コンテナ

J2EE のアプリケーションモデルでは、エンタープライズアプリケーションを 3 つの基本部品 (コンポーネント、コンテナ、およびコネクタ) に分割します。アプリケーション開発者が主に扱うのはコンポーネントであり、システムベンダーは、複雑さを隠蔽し移植性を促進するコンテナおよびコネクタを実装します。コンテナはクライアントとコンポーネントの間を仲介し、トランザクションサポートやリソース

プーリングなどのサービスをその両者に透過的に提供します。コンテナの仲介により、コンポーネントの動作の多くを、プログラムコードで指定するのではなく配備時に指定することが可能になります。

Application Server では、各サーバーインスタンスには 2 種類のコンテナ (Web コンテナと EJB コンテナ) が含まれます。コンテナは、J2EE コンポーネントのセキュリティーやトランザクション管理などのサービスを提供する実行時環境です。JavaServer Pages (JSP) やサーブレットなどの Web コンポーネントは、Web コンテナ内で実行されます。Enterprise JavaBeans は EJB コンテナ内で実行されます。

J2EE サービス

J2EE プラットフォームのサービスは、アプリケーションプログラミングを簡素化します。また、配備環境で利用可能なリソースを使用するために、コンポーネントやアプリケーションを配備時にカスタマイズすることを可能にします。この節では、J2EE プラットフォームのネームサービス、配備サービス、トランザクションサービス、およびセキュリティーサービスの概要を示します。J2EE プラットフォームは、次のようなサービスをアプリケーションに対して提供します。

- ネーミング -ネームサービスとディレクトリサービスは、オブジェクトを名前にバインドします。J2EE アプリケーションは、オブジェクトの Java Naming and Directory Interface (JNDI) 名を検索することによってオブジェクトを検出できます。
- セキュリティー - Java Authorization Contract for Containers (JACC) は、J2EE コンテナに関して定義された一連のセキュリティー規約です。クライアントの ID に基づいて、コンテナはコンテナのリソースおよびサービスに対するアクセスを制限できます。
- トランザクション管理 - トランザクションは作業の分割不能な単位です。たとえば、銀行口座間での資金の振り替えがトランザクションにあたります。トランザクション管理サービスは、トランザクションが完了するか、またはロールバックされるかの二者択一性を保証します。
- メッセージサービス - 別々のシステム上でホストされたアプリケーションどうしが、Java™ Message Service (JMS) を利用してメッセージを交換することによって互いに通信できます。JMS は J2EE プラットフォームの根幹的な部分であり、異機種システム混在のエンタープライズアプリケーションを統合する作業を簡略化します。

Web サービス

クライアントは HTTP、RMI/IIOP、JMS 経由でのアクセスに加えて、リモート Web サービスとして J2EE 1.4 アプリケーションにアクセスできます。Web サービスは、Java API for XML-based RPC (JAX-RPC) を使用して実装されます。J2EE アプリケー

ションは Web サービスに対するクライアントとして動作することもでき、これはネットワークアプリケーションで典型的な構成です。

Web Services Description Language (WSDL) は、Web サービスのインタフェースを記述する XML 形式です。Web サービスのコンシューマは、WSDL ドキュメントを動的に解析して、Web サービスが提供する操作とその実行方法を特定できます。Application Server では、ほかのアプリケーションが Java API for XML Registries (JAXR) を経由してアクセス可能なレジストリを使用して、Web サービスインタフェースの記述を分散させています。

クライアントアクセス

クライアントは複数の方法で J2EE アプリケーションにアクセスできます。ブラウザクライアントは、ハイパーテキストトランスファープロトコル (HTTP) を使用して Web アプリケーションにアクセスします。セキュリティー保護された通信のために、ブラウザでは、Secure Sockets Layer (SSL) を使用する HTTPS プロトコルを利用します。

アプリケーションクライアントコンテナ内で動作するリッチクライアントアプリケーションは、オブジェクトリクエストブローカ (ORB)、リモートメソッド呼び出し (RMI) および (IIOP) internet inter-ORB protocol、または IIOP/SSL (セキュリティー保護された IIOP) を使用して Enterprise JavaBeans (EJB) を直接検索し、アクセスできます。そのようなアプリケーションは、HTTP/HTTPS、JMS、および JAX-RPC を使用してアプリケーションや Web サービスにアクセスできます。それらのアプリケーションでは JMS を使用して、アプリケーションおよびメッセージ駆動型 Bean との間でメッセージを送受信します。

J2EE Web サービスにアクセスできるのは、WS-I Basic Profile (Web サービス相互運用性基本プロファイル) に準拠したクライアントです。WS-I は J2EE 標準の根幹的な部分であり、Web サービスの相互運用性について定義しています。WS-I に準拠することにより、サポートされているすべての言語で記述されたクライアントが、Application Server に配備された Web サービスにアクセスできます。

最適なアクセス機構は、個別のアプリケーションおよび予想されるトラフィック量によって異なります。Application Server は、HTTP、HTTPS、JMS、IIOP、および IIOP/SSL のそれぞれに対応した設定が可能なリスナーをサポートします。各プロトコルに対して複数のリスナーを設定し、スケーラビリティーと信頼性を高めることができます。

J2EE アプリケーションは、ほかのサーバー上に配備された EJB モジュールなどの J2EE コンポーネントのクライアントとしても動作でき、これらのアクセス機構のうち任意のものを使用できます。

外部システムとリソース

J2EE プラットフォームでは、外部のシステムのことを「リソース」と呼びます。たとえば、データベース管理システムは JDBC リソースです。各リソースは、その Java Naming and Directory Interface (JNDI) 名によって一意に識別されます。アプリケーションは、次の API とコンポーネントを通して外部システムにアクセスします。

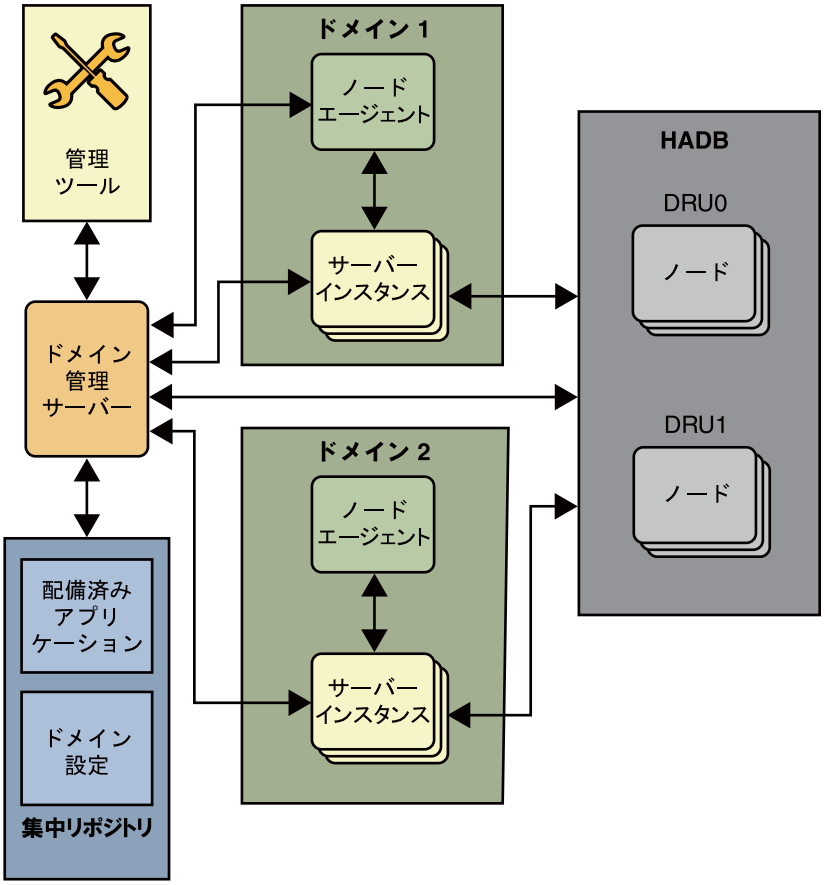
- *Java Database Connectivity (JDBC)* - データベース管理システム (DBMS) は、データを格納、組織化、および取得するための機能を提供します。大部分のビジネスアプリケーションは、アプリケーションが JDBC 経由でアクセスするリレーショナルデータベースにデータを格納します。Application Server に含まれる PointBase DBMS は、サンプルアプリケーションでの使用、アプリケーション開発、プロトタイプ作成などの用途に適していますが、配備用途には適していません。Application Server では、各種の主要リレーショナルデータベースに接続するための、動作確認済み JDBC ドライバを提供しています。これらのドライバは配備に適しています。
- Java Message Service (JMS) - メッセージングは、ソフトウェアコンポーネントまたはアプリケーション間で通信を行うための手段です。メッセージングクライアントは、Java Messaging Service (JMS) API を実装するメッセージングプロバイダを介して、任意のほかのクライアントとの間でメッセージを送受信します。Application Server には、高性能な JMS ブローカーである Sun Java System Message Queue が含まれています。Application Server Platform Edition には、Message Queue の無償版である Platform Edition が含まれます。Application Server Enterprise Edition には、クラスタ化とフェイルオーバーをサポートする Message Queue Enterprise Edition が含まれます。
- **J2EE** コネクタ - J2EE コネクタアーキテクチャーは、J2EE アプリケーションと既存の企業情報システム (EIS) との統合を可能にします。アプリケーションは、「コネクタ」または「リソースアダプタ」と呼ばれる、移植性のある J2EE コンポーネントを介して、JDBC ドライバを使用した RDBMS へのアクセスと同様の方法で EIS にアクセスします。リソースアダプタは、スタンドアロンのリソースアダプタアーカイブ (RAR) モジュールとして配布されるか、または J2EE アプリケーションアーカイブに組み込まれます。RAR 形式のリソースアダプタは、ほかの J2EE コンポーネントと同様に配備されます。Application Server には、一般的な EIS との統合が可能な評価用のリソースアダプタが含まれています。
- **JavaMail** - アプリケーションは JavaMail API を介して Simple Mail Transport Protocol (SMTP) サーバーにアクセスし、電子メールを送受信できます。

Application Server のコンポーネント

この節では、Sun Java System Application Server のコンポーネントについて説明します。

次の図は、これらの Application Server コンポーネントが、高可用性を提供する単純なサンプルトポロジを使用して相互に作用するしくみを例示したものです。このサンプルトポロジでは、クラスタとして編成された 2 台のマシンを 1 人の管理者が管理します。HADB プロセスと Application Server プロセスは同じマシン上に位置します。ドメイン管理サーバーは、独立したマシンに単独で配置することも、アプリケーションサーバーインスタンスをホストしているいずれかのマシンに配置することもできます。図中の線は通信または制御を示します。

まず、ブラウザベースの管理コンソールなどの管理ツールがドメイン管理サーバー (DAS) と通信し、その後 DAS がノードエージェントまたはサーバーインスタンスと通信します。



サーバーインスタンス

サーバーインスタンスは、単一の Java 仮想マシン (JVM) プロセス内で実行される Application Server です。Application Server は、Java 2 Standard Edition (J2SE) 5.0 および 1.4 での動作が検証されています。推奨される J2EE 配布は Application Server のインストールに含まれています。

Application Server と付属の JVM はともに、マルチプロセッサ構成への拡張が可能なように設計されているため、通常は 1 台のマシン上に 1 つのサーバーインスタンスを作成すれば十分です。ただし、アプリケーションの遮断と順次アップグレードのために、1 台のマシン上に複数のインスタンスを作成すると有利な場合もあります。場合によっては、複数のインスタンスが動作する 1 台の大規模サーバーを複数の管理ドメインで使用できます。管理ツールを使用することで、複数のマシンにまたがってサーバーインスタンスを容易に作成、削除、および管理できます。

管理ドメイン

「管理ドメイン」(または単に「ドメイン」) は、まとめて管理されるサーバーインスタンスのグループです。1 つのサーバーインスタンスは単一の管理ドメインに属します。ドメイン内のインスタンスは、複数の異なる物理ホスト上で実行できます。

Application Server の 1 つのインストールから複数のドメインを作成できます。サーバーインスタンスをドメインにグループ化することにより、さまざまな組織および管理者が 1 つの Application Server インストールを共有できます。各ドメインには、固有の設定、ログファイル、およびアプリケーションの配備領域があり、これらはほかのドメインとは無関係です。あるドメインの構成を変更しても、ほかのドメインの構成には影響しません。同様に、あるドメインにアプリケーションを配備しても、そのアプリケーションがほかのドメインに配備されたり、ほかのドメインから認識可能になることはありません。管理者は常に 1 つのドメインに対する認証しか受けられず、そのドメイン上での管理しか実行できません。

Domain Administration Server (DAS)

ドメインには 1 つのドメイン管理サーバー (DAS) があります。これは、管理アプリケーションのホストとなる、特別に設計されたアプリケーションサーバーインスタンスです。DAS は管理者を認証し、管理ツールからの要求を受け付け、ドメイン内のサーバーインスタンスと通信して要求を実行します。

管理ツールには、コマンド行ユーティリティーの `asadmin` と、ブラウザベースの管理コンソールがあります。Application Server では、サーバー管理のための JMX ベースの API も提供されます。管理者が同時に表示および管理できるのは 1 つのドメインのみであり、これによってセキュリティーで保護された分離が実現されています。

DAS のことを「管理サーバー」または「デフォルトサーバー」と呼ぶ場合もあります。DAS をデフォルトサーバーと呼ぶ理由は、一部の管理操作のデフォルトターゲットであるためです。

DAS はアプリケーションサーバーインスタンスであるため、テスト目的で J2EE アプリケーションをホストすることもできます。ただし、本番アプリケーションのホストを目的として DAS を使用しないでください。たとえば、本番アプリケーションをホストする予定のクラスタやインスタンスをまだ作成していない場合に、アプリケーションを DAS に配備することができます。

DAS は、各ドメインおよびすべての配備済みアプリケーションの設定を格納するリポジトリを保持します。DAS がアクティブでないか停止している場合、アクティブなサーバーインスタンスのパフォーマンスまたは可用性への影響はありませんが、管理上の変更は実行できません。状況によっては、セキュリティー上の理由から、本番構成を凍結することなどを目的として意図的に DAS プロセスを停止すると便利な場合があります。

ドメイン設定やアプリケーションをバックアップおよび復元するための管理コマンドが用意されています。標準のバックアップ手順および復元手順を使用して、作業中の構成を迅速に復元できます。DAS ホストで障害が発生した場合、以前のドメイン設定を復元するために新しい DAS インストールを作成する必要があります。手順については、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』の「ドメイン管理サーバーの再作成」を参照してください。

Sun Cluster Data Services は、DAS ホストの IP アドレスのフェイルオーバーと、グローバルファイルシステムの使用によって高可用性を実現します。このソリューションにより、多くの種類の障害に対してほとんど停止しないレベルの可用性が、DAS およびリポジトリに対して提供されます。Sun Cluster Data Services は、Sun Java Enterprise System に付属するか、または Sun Cluster とともに別途購入する形で提供されます。詳細は、Sun Cluster Data Services のマニュアルを参照してください。

クラスタ

クラスタとは、同じアプリケーション、リソース、および設定情報を共有するサーバーインスタンスの集合に名前を付けたものです。異なるマシン上のサーバーインスタンスを 1 つの論理クラスタにまとめ、それらのインスタンスを 1 つの単位として管理できます。マルチマシンクラスタのライフサイクルは、DAS を使用して容易に制御できます。

クラスタにより、水平方向のスケーラビリティー、負荷分散、およびフェイルオーバー保護が使用可能になります。定義により、クラスタ内のすべてのインスタンスに対してリソースとアプリケーションの設定は同じになります。あるサーバーインスタンスまたはクラスタ内のあるマシンに障害が起きると、ロードバランサは障害を検出し、障害の起きたインスタンスからクラスタ内の他のインスタンスにトラフィックをリダイレクトし、ユーザーセッションの状態を回復します。クラスタ内のすべてのインスタンス上には同一のアプリケーションとリソースがあるため、インスタンスはクラスタ内のほかのどのインスタンスにも処理を継続させることができます。

クラスタ、ドメイン、およびインスタンスの関係は次のようになります。

- 1 つの管理ドメイン内に任意の数 (0 個を含む) のクラスタを作成できます。
- 1 つのクラスタは 1 つ以上のサーバーインスタンスで構成できます。
- 1 つのクラスタは単一のドメインに属します。

ノードエージェント

ノードエージェントは、DAS をホストするマシンを含め、サーバーインスタンスをホストするすべてのマシン上で実行される軽量プロセスです。ノードエージェントは次の機能を実行します。

- DAS の指示に従い、サーバーインスタンスを起動および停止します。

- 障害の発生したサーバーインスタンスを再起動します。
- 障害が発生したサーバーのログファイルのビューを提供し、リモート診断を支援します。
- DAS がその監視下のサーバーインスタンスを起動するときに、各サーバーインスタンスのローカル設定リポジトリを DAS の集中リポジトリと同期します。
- インスタンスが最初に作成されるとき、インスタンスに必要なディレクトリを作成し、インスタンスの設定を集中リポジトリと同期します。
- サーバーインスタンスが削除されるときに適切なクリーンアップを実行します。

各物理ホストには、ホストが属する各ドメインに対して少なくとも 1 つのノードエージェントが必要です。1 台の物理ホストに、複数のドメインに属するインスタンスを配置する場合、そのホストにはそれぞれのドメインに対するノードエージェントが必要です。

ノードエージェントはサーバーインスタンスを起動および停止するため、常に実行中である必要があります。したがって、ノードエージェントはオペレーティングシステムの起動時に起動されます。Solaris およびその他の Unix プラットフォームでは、ノードエージェントを `inetd` プロセスによって起動できます。Windows では、ノードエージェントを Windows のサービスにすることができます。

ノードエージェントの詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 8 章「ノードエージェントの設定」を参照してください。

名前付き設定

「名前付き設定」は、Application Server のプロパティー設定をカプセル化する抽象化オブジェクトです。クラスタおよびスタンドアロンのサーバーインスタンスは、名前付き設定を参照してそれぞれのプロパティー設定を取得します。名前付き設定を使用することにより、IP アドレス、ポート番号、ヒープメモリー容量など一部の設定を除く J2EE コンテナの設定が、そのコンテナが位置する物理マシンから分離されます。名前付き設定を使用することにより、Application Server の管理をより強力に、また柔軟に行えるようになります。

設定変更を適用するには、名前付き設定のプロパティー設定を変更します。変更すると、その設定を参照するすべてのクラスタおよびスタンドアロンインスタンスが変更内容を取得します。名前付き設定の削除は、その設定へのすべての参照が削除済みの場合にしか行えません。1 つのドメインに複数の名前付き設定を含めることができます。

Application Server には、`default-config` という名称のデフォルト設定が付属します。デフォルト設定は、Application Server Platform Edition の場合は開発者の生産性を高めるように、Standard Edition および Enterprise Edition の場合はセキュリティーと可用性を高めるように最適化されています。

デフォルト設定をベースにして独自の名前付き設定を作成し、その設定を目的に合わせてカスタマイズできます。名前付き設定の作成と管理には、管理コンソールおよび asadmin コマンド行ユーティリティーを使用します。

HTTP ロードバランサプラグイン

ロードバランサは、複数の物理マシン間で作業負荷を分散させることによって、システムの全体的なスループットを向上させます。Application Server Enterprise Edition には、Sun Java System Web Server、Apache Web Server、および Microsoft Internet Information Server 用のロードバランサプラグインが含まれます。

ロードバランサプラグインは、HTTP および HTTPS 要求を受け付け、それをクラスタ内のアプリケーションサーバーインスタンスのうちの 1 つに転送します。(ネットワーク障害により) インスタンスが停止して使用不能または応答不能になると、要求は既存の使用可能なマシンにリダイレクトされます。ロードバランサはまた、失敗したインスタンスが復旧したことを認識し、それに応じて負荷を再配分することもできます。

状態を持たない単純なアプリケーションであれば、負荷分散されたクラスタで十分なこともあります。しかし、セッション状態を持ったミッションクリティカルなアプリケーションの場合は、負荷分散されたクラスタを HADB とともに使用します。

システムで負荷分散を設定するには、Application Server に加えて、Web サーバーおよびロードバランサプラグインをインストールする必要があります。その後、次のことを行う必要があります。

- 負荷分散に参加させる Application Server クラスタを作成します。
- 負荷分散用に設定したこれらのクラスタにアプリケーションを配備します。

負荷分散に関わるサーバーインスタンスとクラスタは、同種の環境を確保しています。これは、通常、サーバーインスタンスが同じサーバー設定を参照し、同じ物理リソースにアクセスでき、さらに配備された同じアプリケーションを持っていることを意味します。この均質性によって、障害の前後に、ロードバランサが常に負荷を均等にクラスタ内のアクティブなインスタンスに分散することが保証されます。

asadmin コマンド行ツールを使用して、ロードバランサ設定を作成し、クラスタおよびサーバーインスタンスへの参照をその設定に追加し、クラスタでロードバランサによる参照を有効にし、アプリケーションで負荷分散を有効にし、必要に応じて健全性検査を作成し、ロードバランサ設定ファイルを生成し、最後に、ロードバランサ設定ファイルを Web サーバーの config ディレクトリにコピーします。管理者は、スクリプトを作成してこのプロセス全体を自動化できます。

詳細および完全な設定手順については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 5 章「HTTP 負荷分散の設定」を参照してください。

セッション持続性

J2EE アプリケーションは一般に、大量のセッション状態データを保持しています。Web ショッピングカートは、セッション状態の古典的な例です。アプリケーションはまた、頻繁に必要になるデータをセッションオブジェクトにキャッシュすることもできます。実際、ユーザーとの対話が多いほぼすべてのアプリケーションには、セッション状態の保持が必要になります。HTTP セッションとステートフルセッション Bean (SFSB) の両方がセッション状態データを持ちます。

セッション状態はデータベースに格納されるトランザクション状態ほど重要ではありませんが、サーバー障害の前後でセッション状態を保持することがエンドユーザーにとって重要な場合があります。Application Server は、このセッション状態をリポジトリに保存する (持続する) ための機能を提供します。ユーザーセッションをホストするアプリケーションサーバーインスタンスで障害が発生した場合、そのセッション状態の復元が可能です。情報を失うことなくそのセッションを継続できます。

Application Server は、次のタイプのセッション持続性ストアをサポートします。

- メモリー
- 高可用性 (HA)
- ファイル

メモリーを使用した持続性では、状態は常にメモリー内に保持され、障害が発生すると情報は失われます。HA 形式の持続性では、Application Server は HTTP セッションと SFSB セッションの両方で HADB を持続性ストアとして使用します。ファイルを使用した持続性では、Application Server はセッションオブジェクトを直列化し、それらのオブジェクトを、セッションマネージャーのプロパティーで指定されたファイルシステム上の位置に格納します。SFSB に関しては、HA 形式の持続性を指定しない場合、Application Server はこの位置の session-store サブディレクトリに状態情報を格納します。

保存が必要な変更がないかどうか、SFSB の状態をチェックする処理のことを「チェックポイント設定」と呼びます。チェックポイント設定を有効にした場合、この処理は通常、トランザクションがロールバックした場合を含め、SFSB が関係するすべてのトランザクションの完了後に発生します。ステートフルセッション Bean 開発の詳細は、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide』の「Using Session Beans」を参照してください。SFSB フェイルオーバーの有効化の詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「ステートフルセッション Bean のフェイルオーバー」を参照してください。

Application Server がサービスを提供している要求の数とは別に、セッション持続性の設定は、各要求内のセッション情報だけでなく、HADB が 1 分あたりに受信する要求の数にも影響します。

セッション持続性の設定の詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 9 章「高可用性 (HA) セッション持続性とフェイルオーバーの設定」を参照してください。

クラスタ内の IIOP 負荷分散

IIOP 負荷分散は、IIOP クライアントの要求を複数の異なるサーバーインスタンスまたはネームサーバーに分散します。目標は、負荷をクラスタ間に均等に拡散して、スケーラビリティーを実現することです。IIOP 負荷分散と、EJB クラスタ化および Sun Java System Application Server の 可用性機能を組み合わせることにより、負荷分散に加えて EJB フェイルオーバーも実現されます。

クライアントがオブジェクトに対して JNDI 検索を実行すると、ネームサービスは、特定のサーバーインスタンスに関連付けられた `InitialContext` (IC) オブジェクトを作成します。それ以降、その IC オブジェクトを使用して作成された検索要求はすべて、同じサーバーインスタンスに送信されます。その `InitialContext` を使用して検索された `EJBHome` オブジェクトはすべて、同じターゲットサーバーにホストされます。また、それ以降に取得された Bean 参照もすべて、同じターゲットホスト上に作成されます。`InitialContext` オブジェクトの作成時に、ライブターゲットサーバーのリストがすべてのクライアントによってランダムに選択されるため、これにより負荷分散が効果的に実現されます。ターゲットサーバーインスタンスが停止すると、検索または EJB メソッド呼び出しは、別のサーバーインスタンスに処理が引き継がれます。

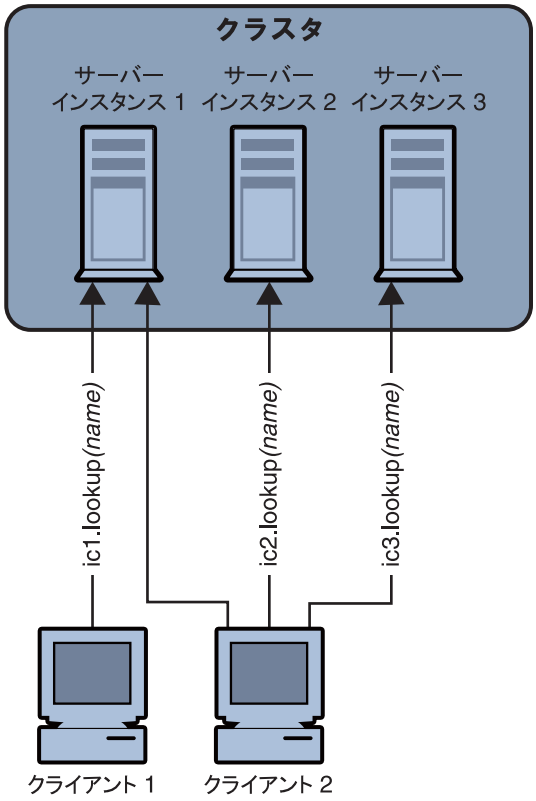


図 1–1　クラスタ内の IIOP 負荷分散

たとえば、この図で示されている ic1、ic2、および ic3 は、クライアント 2 のコードで作成される 3 つの異なる `InitialContext` インスタンスです。それらはクラスタ内の 3 つのサーバーインスタンスに分散されます。そのため、このクライアントによって作成される EJB は 3 つのインスタンス間に分散されます。クライアント 1 が作成した InitialContext オブジェクトは 1 つだけであり、このクライアントからの Bean 参照はサーバーインスタンス 1 のみが対象です。サーバーインスタンス 2 が停止した場合、ic2 に対する検索要求は別のサーバーインスタンス (サーバーインスタンス 3 とは限らない) にフェイルオーバーされます。以前にサーバーインスタンス 2 でホストされていた Bean に対するすべての Bean メソッド呼び出しも、そのように処理するのが安全であれば、別のインスタンスに自動的にリダイレクトされます。検索のフェイルオーバーは自動的ですが、EJB モジュールは再試行が安全なときに限りメソッド呼び出しを再試行します。

RMI-IIOP 負荷分散とフェイルオーバーは、透過的に発生します。アプリケーションの配備中に、特別な手順は必要ありません。クラスタに新しいインスタンスを追加したり削除したりしても、そのクラスタに関する既存のクライアントの表示は更新されません。クライアント側で、端点の一覧を手動で更新する必要があります。

Message Queue と JMS リソース

Sun Java System Message Queue (MQ) は、分散アプリケーションのための、信頼性のある非同期メッセージングを提供します。MQ は Java Message Service (JMS) 標準を実装するエンタープライズメッセージングシステムです。MQ は、メッセージ駆動型 Bean (MDB) などの J2EE アプリケーションコンポーネントに対してメッセージングを提供します。

Application Server は Sun Java System Message Queue を Application Server に統合することにより Java Message Service (JMS) API を実装します。Application Server Enterprise Edition には、フェイルオーバー、クラスタ化、および負荷分散の各機能を持つ Enterprise バージョンの MQ が含まれます。

基本的な JMS 管理タスクには、Application Server の管理コンソールおよび `asadmin` コマンド行ユーティリティーを使用します。

Message Queue クラスタの管理など、高度なタスクの場合は、*install_dir*`/imq/bin` ディレクトリに用意されたツールを使用します。Message Queue の管理の詳細は、『Sun Java System Message Queue 管理ガイド』を参照してください。

メッセージフェイルオーバーのための JMS アプリケーションの配備および MQ クラスタ化の詳細は、65 ページの「Message Queue ブローカの配備の計画」を参照してください。

高可用性データベース

この節の内容は次のとおりです。

31 ページの「セッション持続性」では、J2EE アプリケーションでのセッション持続性の必要性について説明しました。Application Server では、高可用性セッションストアとして高可用性データベース (HADB) を使用します。HADB は Application Server Enterprise Edition に含まれていますが、配備では独立したホスト上で実行できます。HADB は、HTTP セッションおよびステートフルセッション Bean データの高可用性データストアを提供します。

この分離型アーキテクチャーには、次のような利点があります。

- 高可用性クラスタ内のサーバーインスタンスを疎結合し、高性能の J2EE コンテナとして機能させることができます。
- サーバーインスタンスの停止および起動が、ほかのサーバーやその可用性に影響を及ぼしません。

- HADB は、たとえばシングルプロセッサやデュアルプロセッサの、比較的安価なマシンの異なるセット上で実行できます。複数のクラスタがこれらのマシンを共有できます。配備のニーズに応じて、Application Server と同じマシン上で HADB を実行する (共存) か、異なるマシン上で実行する (独立層) かを選択できます。この 2 つの選択肢の詳細は、75 ページの「共存トポロジ」を参照してください。
- 状態管理の要件の変化に合わせて、既存のクラスタやその内部のアプリケーションに影響を及ぼすことなく、HADB システムにリソースを追加できます。

注 – HADB は Application Server による使用のために最適化されており、汎用データベースなどのアプリケーションによる使用は想定されていません。

HADB のシステム要件

HADB ホストのシステム要件は次のとおりです。

- 各 HADB ノードに少なくとも 1 個の CPU
- 各ノードに最低 512M バイトのメモリー

ネットワーク構成の要件については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 2 章「高可用性 (HA) データベースのインストールと設定」を参照してください。さらに高度な可用性を実現するための追加要件については、39 ページの「二重障害の防止」を参照してください。

HADB のアーキテクチャー

HADB は、「ノード」のペアで構成される分散システムです。36 ページの「データ冗長ユニット」で示すように、ノードは 2 つのデータ冗長ユニット (DRU) に分割され、1 つのノードは各 DRU 内の各ペアに属します。

各ノードは次の要素で構成されます。

- トランザクション状態レプリケーションのためのプロセスのセット
- プロセス間の通信に使用される共有メモリーの専用領域
- 1 つ以上の二次記憶装置 (ディスク)

HADB ノードの 1 つのセットが、1 つ以上の「セッションデータベース」をホストすることができます。各セッションデータベースは、個別のアプリケーションサーバークラスタと関連付けられます。クラスタを削除すると、関連付けられたセッションデータベースも削除されます。

ノードとノードプロセス

HADB ノードには、次の 2 種類があります。

- データを格納する「アクティブノード」。

- 最初はデータを含まないが、アクティブノードが使用不能になった場合にアクティブノードの役割を果たす「スペアノード」。スペアノードの使用は任意ですが、より高い可用性を達成するために役立ちます。

各ノードには、1 つの親プロセスと複数の子プロセスがあります。ノードスーパーバイザー (NSUP) と呼ばれる親プロセスは、管理エージェントによって起動されます。このプロセスは、子プロセスの作成とその実行の継続の役割を果たします。

子プロセスには次のものがあります。

- トランザクションサーバープロセス (TRANS)。分散ノード上のトランザクションを調整し、データ記憶領域を管理します。
- 関係代数サーバープロセス (RELALG)。ソートや AND 結合などの複雑な関係代数クエリーを調整および実行します。
- SQL 共有メモリーサーバープロセス (SQLSHM)。SQL ディレクトリキャッシュを保守します。
- SQL サーバープロセス (SQLC)。クライアントのクエリーを受信し、それらをローカル HADB の命令にコンパイルし、命令を TRANS に送信し、結果を受け取ってクライアントに伝達します。各ノードには 1 つのメイン SQL サーバーと、個々のクライアント接続用に 1 つのサブサーバーがあります。
- ノードマネージャーサーバープロセス (NOMAN)。hadbm 管理クライアントによって発行された管理コマンドを実行するために管理エージェントが使用します。

データ冗長ユニット

すでに説明したように、1 つの HADB インスタンスには DRU のペアが 1 つ含まれます。各 DRU には、もう一対の DRU と同数のアクティブノードとスペアノードが存在します。DRU 内の各アクティブノードに対し、もう一方の DRU 内に「ミラーノード」が存在します。ミラーリングにより、各 DRU がデータベースの完全なコピーを保持します。

次の図は、6 つのノードが存在するサンプルの HADB アーキテクチャーを示したものです。4 つのアクティブノードと 2 つのスペアノードが存在します。ノード 0 と 1 はミラーペアであり、ノード 2 と 3 も同様です。この例では、各ホストに 1 つのノードが存在します。一般に、ホストに十分なシステムリソースがあれば、複数のノードを 1 つのホストに共存させることができます (35 ページの「HADB のシステム要件」を参照)。

注 – HADB ノードをホストするマシンは、ペアで (各 DRU に 1 台ずつ) 追加する必要があります。

HADB では、データおよびサービスをレプリケートすることによって高可用性を実

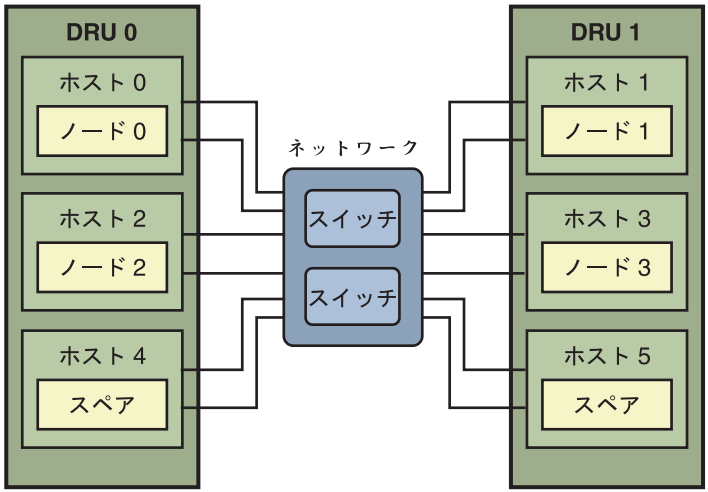


図 1–2　二重相互接続を持つサンプル HADB 構成

現します。ミラーノード上のデータレプリカは、「プライマリレプリカ」および「ホットスタンバイレプリカ」として指定されます。プライマリレプリカは、挿入、削除、更新、読み取りなどの操作を実行します。ホットスタンバイレプリカは、プライマリレプリカの操作のログレコードを受信し、トランザクションの寿命内にそれらの操作を再実行します。読み取り操作はプライマリレプリカによってのみ実行されるため、ログに記録されません。各ノードにはプライマリレプリカとホットスタンバイレプリカの両方が含まれ、同じ役割を果たします。データベースは、DRU 内のアクティブノード間で断片化および分散されます。ミラーペア内の各ノードには、データフラグメントの同じ集合が含まれます。ミラーノードにデータを複製することを「レプリケーション」と呼びます。レプリケーションにより、HADB で高可用性を提供できます。あるノードで障害が発生した場合、ほとんど即時 (数秒以内) にそのミラーノードへの引き継ぎが行われます。レプリケーションにより可用性が保証され、データやサービスを失うことなくノードまたは DRU の障害が隠蔽されます。

障害が発生したノードの機能を引き継ぐとき、ミラーノードでは 2 つの作業を実行する必要があります。ミラーノード自体の作業と、障害が発生したノードの作業です。ミラーノードに十分なリソースがない場合、過負荷によってミラーノードのパフォーマンスが低下し、ミラーノードの障害の可能性も高まります。あるノードで障害が発生すると、HADB はそのノードの再起動を試みます。ハードウェアの故障などが原因で、障害が発生したノードが再起動しない場合、システムは動作を続けますが可用性は低下します。

HADB は 1 つのノード、DRU 全体、または複数のノードの障害に対する耐性を備えますが、ノードおよびそのミラーで障害が発生したときの「二重障害」への耐性はありません。二重障害の可能性を下げる方法の詳細は、39 ページの「二重障害の防止」を参照してください。

スペアノード

あるノードで障害が発生すると、そのミラーノードが元のノードの役割を引き継ぎます。障害が発生したノードにスペアノードがない場合、この時点で、障害が発生したノードにミラーがないことになります。スペアノードは、障害が発生したノードのミラーを自動的に置き換えます。スペアノードを用意しておくことにより、ミラーノードのない状態でシステムを運用する時間を短縮できます。

スペアノードは通常時はデータを保持しませんが、DRU 内のアクティブノードの障害を常に監視しています。ノードで障害が発生し、指定された期間内に復旧しない場合、スペアノードはミラーノードからデータをコピーし、ミラーノードとの同期を行います。この処理にかかる時間は、コピーされるデータの量と、システムおよびネットワークの能力によって異なります。同期のあとで、スペアノードは手動での作業を必要とすることなく自動的にミラーノードを代替します。これによりミラーノードの過負荷を防ぎ、ミラーへの負荷を分散します。この処理のことを「フェイルバック」または「自己修復」と呼びます。

障害が発生したホストが、ハードウェアの切り替えまたはソフトウェアのアップグレードによって復旧および再起動した場合、元のスペアノードがこの時点でアクティブノードになっているため、そのホストで実行中の 1 つ以上のノードがスペアノードとしてシステムに参加します。

スペアノードは必須ではありませんが、1 台のマシンで障害が発生した場合でも、システムが全体的なサービスレベルを維持できるようにする効果があります。また、スペアノードを用意しておくことにより、アクティブノードのホストマシン上で計画的な保守を実施しやすくなります。スペアマシンの役割を果たす 1 台のマシンを各 DRU に割り当てて、マシンのうち 1 台で障害が発生した場合でも、HADB システムがパフォーマンスと可用性を低下させることなく運用を継続できるようにします。

注 – 原則として、十分な Application Server インスタンス数および HADB ノード数を持つスペアマシンが、使用不能になったマシンを代替するようにします。

スペアノード構成の例

HADB 配備でのスペアノードの使用例を次に示します。使用できる配備トポロジは 2 種類あります。「共存」トポロジでは HADB と Application Server を同じホストに配置し、「独立層」トポロジでは両者を別々のホストに配置します。配備トポロジの詳細は、第 3 章を参照してください。

例: 共存構成

スペアノード構成の例として、4 台の Sun Fire™ V480 サーバーを使用し、各サーバーに 1 つの Application Server インスタンスと 2 つの HADB データノードを配置する共存トポロジを考えます。

スペアノード用に、さらに 2 台のサーバーを割り当てます (各 DRU に 1 台ずつ)。各スペアマシンでは、1 つの Application Server インスタンスと 2 つのスペア HADB ノードを実行します。

例: 独立層構成

HADB 層に 2 台の Sun Fire™ 280R サーバーを配置し、それぞれが 2 つの HADB データノードを実行する個別層トポロジを考えます。1 台のマシンが使用不能になった場合でもこのシステムの完全な能力を維持するには、Application Server インスタンス層と HADB 層のそれぞれに 1 台のスペアマシンを設定します。

Application Server インスタンス層のスペアマシンには、この層内のほかのマシンと同数のインスタンスが必要です。同様に、HADB 層用のスペアマシンには、この層内のほかのマシンと同数の HADB ノードが必要です。

二重障害の防止

HADB の組み込み型のデータレプリケーションにより、単一ノードまたは DRU 全体の障害に対する耐性を HADB に持たせることができます。デフォルトでは、HADB はミラーノードペアまたは両方の DRU で障害が発生した状況を指す「二重障害」を許容できません。そのような場合、HADB は使用不能になります。

前の節で説明したスペアノードの使用に加えて、次の手順に従うことにより、二重障害の可能性を最小化できます。

- 独立した電源装置の用意: 最適な耐障害性のために、1 つの DRU をサポートするサーバーは、(無停電電源装置を通じての) 独立した電力、処理装置、および記憶装置を備える必要があります。1 つの DRU で停電が発生した場合は、電源が復旧するまで、もう一方の DRU 内のノードが引き続き要求を処理します。
- 二重相互接続の実装: 単一のネットワーク障害に対する耐性を持たせるには、図 1–2に示すように、DRU 間でラインおよびスイッチをレプリケートします。

これらの手順は任意ですが、HADB インスタンスの全体的な可用性を高めます。

HADB 管理システム

HADB 管理システムは、組み込み型のセキュリティーを提供し、マルチプラットフォーム管理を促進します。次の図で示すように、HADB 管理アーキテクチャーには次のコンポーネントが含まれます。

- 41 ページの「管理クライアント」
- 42 ページの「管理エージェント」
- 42 ページの「管理ドメイン」
- 43 ページの「リポジトリ」

図で示すように、HADB サービスを実行するすべてのマシン上で 1 つの HADB 管理エージェントが実行されます。各マシンは通常、1 つ以上の HADB ノードをホストします。Application Server ドメインと同様に、HADB 管理ドメインには多数のマシンが含まれます。データベースに耐障害性を持たせるには、ドメイン内に最低 2 台のマシンが必要であり、一般的には、DRU ペアを形成するために偶数のマシンを用意する必要があります。そのため、ドメインには多数の管理エージェントが含まれます。

図で示すように、ドメインには 1 つ以上のデータベースインスタンスを含めることができます。1 台のマシンに、1 つ以上のデータベースインスタンスに属する 1 つ以上のノードを含めることができます。

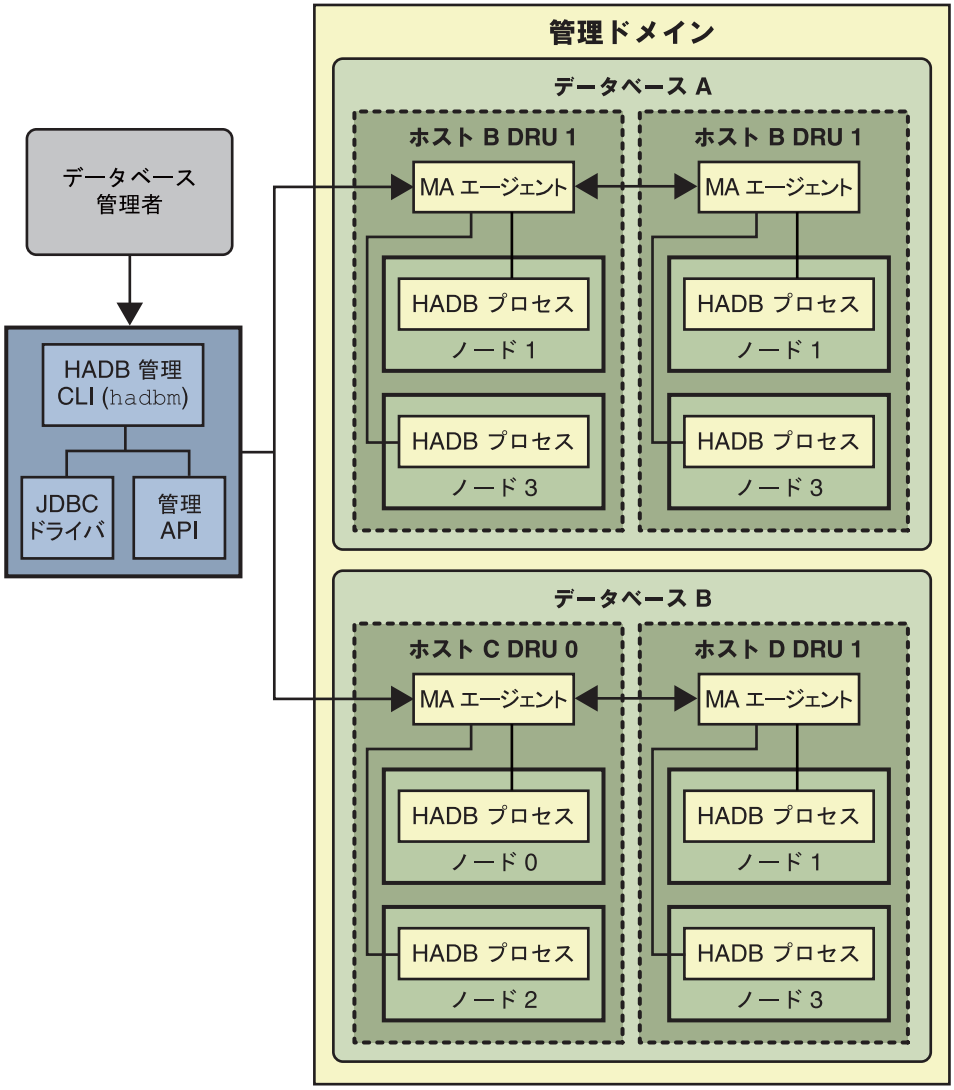


図 1–3　HADB 管理アーキテクチャー

管理クライアント

HADB の「管理クライアント」はコマンド行ユーティリティー `hadbm` であり、HADB ドメインとそのデータベースインスタンスを管理するためのものです。HADB サービスは、関連付けられた Application Server クラスタが停止しているときでも実行を継続できますが、サービスを削除する場合は注意深くサービスをシャットダウンする

必要があります。hadbm の使用方法の詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 3 章「高可用性データベースの管理」を参照してください。

asadmin コマンド行ユーティリティーを使用して、高可用性クラスタと関連付けられた HADB インスタンスを作成および削除できます。詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 9 章「高可用性 (HA) セッション持続性とフェイルオーバーの設定」を参照してください。

管理エージェント

HADB の管理エージェントは、ホスト上のリソースにアクセスできる ma という名前のサーバープロセスです。たとえば、このプロセスはデバイスの作成やデータベースプロセスの起動を実行できます。管理エージェントは、データベースインスタンスの起動や停止などの管理クライアントコマンドを調整および実行します。

管理クライアントは、エージェントのアドレスおよびポート番号を指定することによって管理エージェントに接続します。接続したあとは、管理クライアントは管理エージェントを通して HADB にコマンドを送信します。エージェントは要求を受け取って実行します。したがって、ホストに対して hadbm 管理コマンドを発行する前に、そのホスト上で管理エージェントが実行されている必要があります。管理エージェントは、自動的に起動するシステムサービスとして設定できます。

管理エージェントの可用性の保証

HADB ノードスーパーバイザープロセスが失敗すると、管理エージェントプロセスを再起動して、その可用性を確保します。そのため、配備目的では、HADB の全体的な可用性を維持するために、ma プロセスの可用性を保証する必要があります。再起動のあと、管理エージェントはドメイン内のほかのエージェントからドメインおよびデータベースの設定データを復旧します。

管理エージェントの可用性を保証するには、ホストのオペレーティングシステム (OS) を使用します。Solaris または Linux では、プロセスの障害またはオペレーティングシステムの再起動のあと、init.d が ma プロセスの可用性を保証します。Windows では、管理エージェントは Windows サービスとして動作します。そのため、エージェントで障害が発生した場合や OS が再起動する場合、OS が管理エージェントを再起動します。

管理ドメイン

HADB の管理ドメインはホストの集合であり、各ホストでは同じポート番号で管理エージェントが実行されています。ドメイン内のホストには、1 つ以上の HADB データベースインスタンスを含めることができます。管理ドメインは、エージェントが使用する共通のポート番号と、ドメインを作成したりエージェントをドメインに追加したときに生成される「ドメインキー」と呼ばれる ID によって定義されます。管理エージェントはマルチキャストを使用して通信するため、ドメインの一意 ID を提

供するドメインキーの役割は重要です。Application Server ドメインと一致するように、HADB 管理ドメインを設定できます。

複数のデータベースインスタンスを 1 つのドメインに含めると、複数の開発者グループが個別のデータベースインスタンスを使用できるため、この構成は開発環境で役立ちます。場合によっては、この構成は本番環境でも役立つことがあります。

エージェントの管理操作は、ドメインに属するすべてのエージェント間で協調されます。`hadbm` コマンドを使用してデータベース設定を変更すると、すべてのエージェントがそれに従って設定を変更します。ノードのホスト上の管理エージェントが実行されていない場合、ノードを停止または再起動できません。ただし、一部のエージェントが使用不能な場合でも、HADB 状態または設定変数の値を読み取る `hadbm` コマンドを実行できます。

管理ドメインを操作するには、次の管理クライアントコマンドを使用します。

- **hadbm createdomain**: ホストを指定して管理ドメインを作成します。詳細は、`hadbm-createdomain`(1)を参照してください。
- **hadbm extenddomain**: 既存の管理ドメインにホストを追加します。詳細は、`hadbm-extenddomain`(1)を参照してください。
- **hadbm deletedomain**: 管理ドメインを削除します。詳細は、`hadbm-deletedomain`(1)を参照してください。
- **hadbm reducedomain**: 管理ドメインからホストを削除します。詳細は、`hadbm-reducedomain`(1)を参照してください。
- **hadbm listdomain**: 管理ドメインに定義されたすべてのホストを表示します。詳細は、`hadbm-listdomain`(1)を参照してください。

リポジトリ

管理エージェントは、データベース設定を HADB の「リポジトリ」に格納します。リポジトリはすべての管理エージェント間でレプリケートされるため、高い耐障害性を備えています。サーバー上に設定を保持することにより、管理クライアントがインストールされている任意のコンピュータから管理操作を実行できるようになります。

リポジトリに対して何らかの変更を実行するには、ドメイン内の管理エージェントの大半が実行中である必要があります。したがって、ドメイン内に M 個のエージェントがある場合、リポジトリに変更を行うには、少なくとも M/2 + 1 (端数を切り捨てた整数値) 個のエージェントが実行中である必要があります。

ハードウェアの故障などが原因でドメイン内の一部のホストが使用不能であり、エージェント数不足のため一部の管理コマンドを実行できない場合は、`hadbm disablehost` コマンドを使用して、障害が発生したホストをドメインから削除します。このコマンドの詳細は、`hadbm-disablehost`(1)を参照してください。

セットアップと設定のロードマップ

▼ 高可用性のために Application Server をセットアップおよび設定するには

1. 第 2 章の説明に従って、パフォーマンスおよび QoS の要件と目標を決定します。
2. 第 2 章の 61 ページの「設計上の決定」の説明に従って、システムの規模を決定します。
   - Application Server インスタンスの数
   - HADB ノードおよび HADB ホストの数
   - HADB 記憶領域の容量
3. 第 3 章での説明に従って、システムトポロジを決定します。
   ここでは、Application Server と同じホストマシンまたは異なるマシンのどちらに HADB をインストールするかを決定します。
4. HADB や Web サーバーなどの関連するサブコンポーネントとともに、Application Server インスタンスをインストールします。
5. ドメインとクラスタを作成します。
6. Web サーバーソフトウェアを設定します。
7. ロードバランサプラグインをインストールします。
8. 負荷分散をセットアップおよび設定します。
9. HADB ノードおよび DRU をセットアップおよび設定します。
10. HA セッション持続性のために AS Web コンテナおよび EJB コンテナを設定します。
11. アプリケーションを配備し、高可用性およびセッションフェイルオーバーのために設定します。
12. メッセージを多く利用する場合、フェイルオーバーのために JMS クラスタを設定します。
    詳細は、『Sun Java System Message Queue 3.7 UR1 管理ガイド』の第 9 章「ブローカクラスタを使用した作業」を参照してください。

◆ ◆ ◆ 第 2 章

2 配備の計画

Application Server を配備する前に、まずパフォーマンスと可用性の目標を決定したあと、それに応じてハードウェア、ネットワーク、およびストレージの要件に関する決定を行います。

この章の内容は次のとおりです。

パフォーマンス目標の確立

高パフォーマンスのもっとも単純な意味は、スループットを最大化し、応答時間を短縮することです。これらの基本的な目標を超えて特定の目標を確立するには、次の内容を決定します。

- 配備するアプリケーションやサービスの種類と、クライアントからのアクセス方法。
- 高可用性を必要とするアプリケーションやサービス。
- アプリケーションにセッション状態はあるか、または状態を持たないか。
- システムでサポートする必要のある要求の容量またはスループット。
- システムでサポートする必要のある並行ユーザーの数。
- ユーザー要求に対する許容可能な平均応答時間。
- 要求の間の平均思考時間。

これらのメトリックスの一部は、リモートブラウザエミュレータ (RBE) ツールか、または予測されるアプリケーションアクティビティーのシミュレーションを行う Web サイトのパフォーマンスおよびベンチマークソフトウェアを使用して計算できます。一般に、RBE やベンチマーク製品は並行 HTTP 要求を生成し、次に 1 分あたり

の特定数の要求に対する応答時間を報告します。これらの数値を使用することによって、サーバーアクティビティーを計算できます。

この章で説明されている計算の結果は絶対ではありません。Application Server や独自のアプリケーションのパフォーマンスを微調整するときに照合するための参照点として扱ってください。

この節の内容は次のとおりです。

スループットの見積もり

広義では、スループットは Application Server によって実行された作業の量を測定します。Application Server の場合、スループットは、サーバーインスタンスあたり、1 分あたりに処理された要求の数として定義できます。また、高可用性アプリケーションでは、セッション状態データが定期的に保存されるため、HADB にもスループット要件が課せられます。HADB の場合、スループットは、1 分あたりに格納されるセッションデータの量として定義できます。これは、1 分あたりの HADB 要求の数と、1 要求あたりの平均セッションサイズの積です。

次の節で説明するように、Application Server のスループットは、ユーザー要求の性質やサイズ、ユーザーの数、Application Server インスタンスやバックエンドデータベースのパフォーマンスを含む、多くの要因の関数です。シミュレートされた作業負荷のベンチマークによって、1 台のマシンでのスループットを見積もることができます。

高可用性アプリケーションでは、データが HADB に定期的に保存されるため、追加のオーバーヘッドが発生します。このオーバーヘッドの量は、データの量、データが変更される頻度、およびデータの保存頻度によって異なります。最初の 2 つの要因は対象のアプリケーションによって異なり、さらに後者はサーバー設定によっても影響されます。

HADB のスループットは、1 分あたりの HADB 要求の数に、1 要求あたりの平均データ量を掛けた値として定義できます。HADB のスループットを向上させるには、より多くの HADB ノードと、より大きいストアサイズが必要になります。

Application Server インスタンスへの負荷の見積もり

Application Server インスタンスへの負荷を見積もるには、次の要因を考慮してください。

並行ユーザーの最大数

ユーザーは、Web ブラウザや Java プログラムなどのクライアントを介してアプリケーションと対話します。ユーザーのアクションに基づいて、クライアントは Application Server に要求を定期的に送信します。ユーザーは、そのユーザーのセッションが期限切れでなく、終了されてもいないかぎり、アクティブと見なされます。並行ユーザーの数を見積もる場合は、すべてのアクティブユーザーを含めてください。

次の図は、ユーザーの数に対する、1 分あたりに処理される要求の数 (スループット) の標準的なグラフを示しています。最初は、ユーザーの数が増えると、それに対応してスループットが増加します。ただし、並行要求の数が増えるにつれてサーバーパフォーマンスが飽和し始めるため、スループットは低下し始めます。

並行ユーザーを追加した場合に 1 分あたりに処理できる要求の数が減る点を特定します。この点は、最適なパフォーマンスに達しており、それを超えるとスループットが低下し始める時点を示します。一般には、システムをできるだけ最適なスループットで動作させるようにしてください。追加の負荷を処理し、スループットを向上させるには、処理能力の追加が必要になる場合があります。

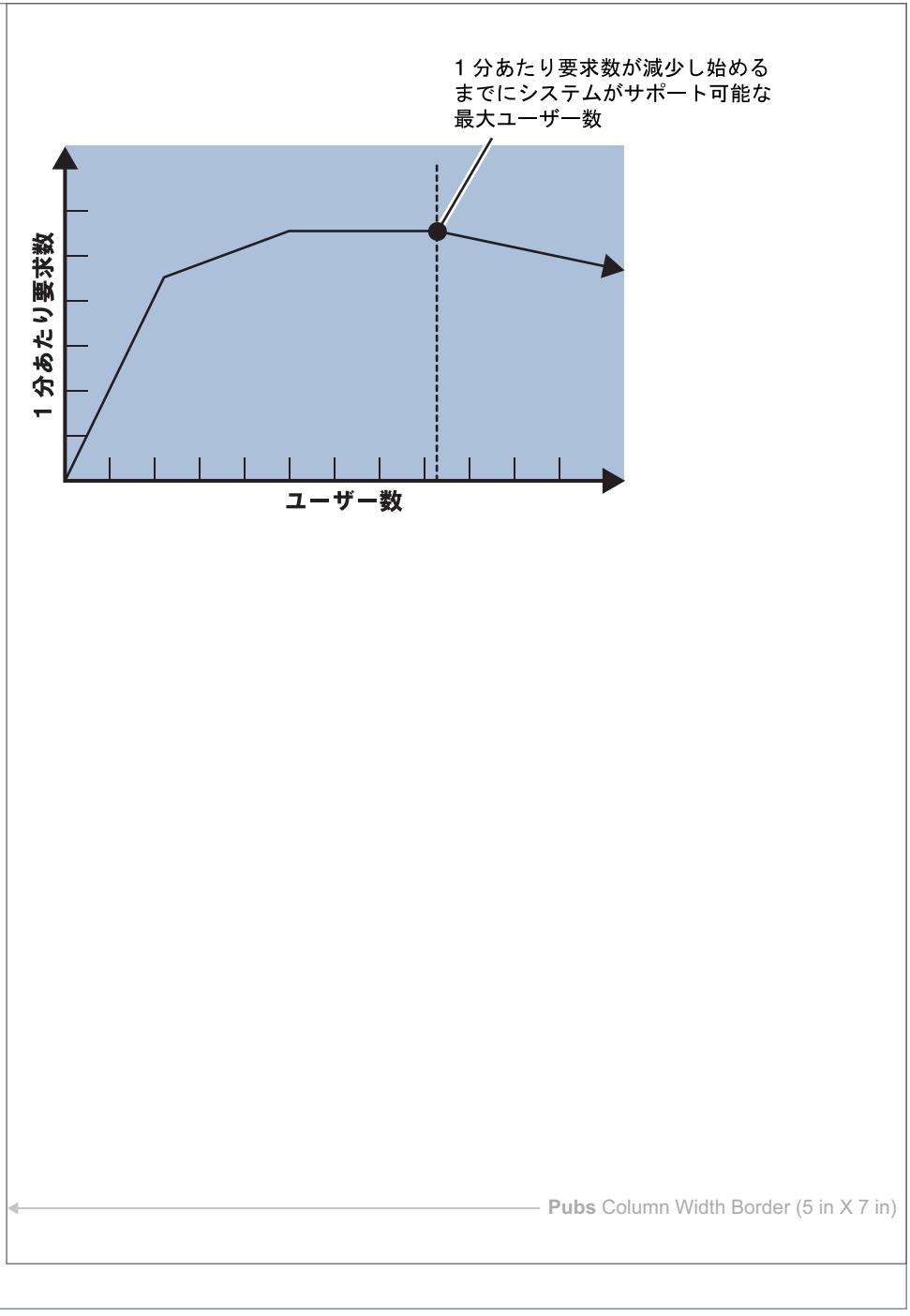


図 2–1 並行ユーザー数に対するスループットの標準的なプロファイル

思考時間

ユーザーが要求を連続して送信することはありません。ユーザーが要求を送信すると、サーバーがその要求を受信し、処理したあと、結果を返します。ユーザーはその時点で、新しい要求を送信する前に一定の時間を費やします。ある要求から次の要求までの時間を、思考時間と呼びます。

思考時間は、ユーザーの種類に依存します。たとえば、Web サービスの場合のようなマシン同士の対話では一般に、思考時間は人間同士の対話より短くなります。思考時間を見積もるために、マシンと人間の対話の混在を考慮することが必要な場合があります。

平均思考時間の特定は重要です。この所要時間を使用すると、1 分あたりに完了する必要のある要求の数や、システムでサポートできる並行ユーザーの数を計算できます。

平均応答時間

応答時間とは、Application Server が、要求の結果をユーザーに返すために費やす時間のことを指します。応答時間は、ネットワーク帯域幅、ユーザーの数、送信される要求の数とタイプ、平均思考時間などの要因によって影響されます。

ここでは、応答時間は平均の応答時間を指します。要求のタイプごとに、独自の最小応答時間があります。ただし、システム性能を評価する場合は、すべての要求の平均応答時間に基づいて分析します。

応答時間が速ければ速いほど、1 分あたりに処理される要求が増えます。ただし、次の図に示すように、システム上のユーザーの数が増えるにつれ、1 分あたりの要求の数が低下するにもかかわらず応答時間も増え始めます。

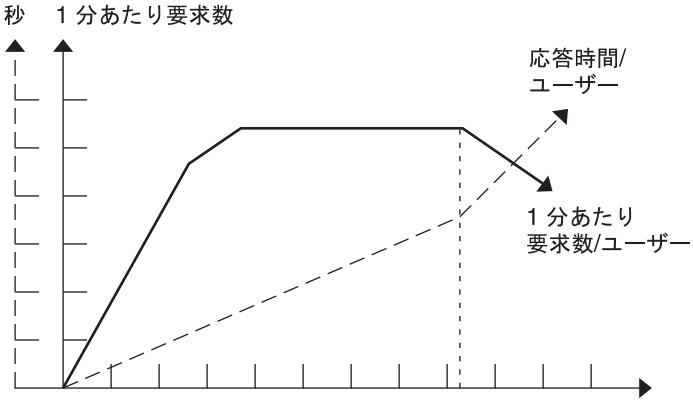


図 2–2 ユーザーの数が増えた場合の応答時間

この図のようなシステム性能のグラフは、特定の点を過ぎると、1 分あたりの要求数が応答時間に反比例することを示しています。点線の矢印で表されているように、1 分あたりの要求数の低下が激しければ激しいほど、応答時間の増加も急になります。

この図の場合、ピーク負荷の点は、1 分あたりの要求数が低下し始める時点になります。この点の前は、数式にピークの数値が使用されていないため、応答時間の計算は必ずしも正確ではありません。この点のあとは、1 分あたりの要求数と応答時間の間に反比例の関係があるため、管理者は、ユーザーの最大数と 1 分あたりの要求数を使用して応答時間をより正確に計算できます。

ピーク負荷時の応答時間 (秒単位) である $T_{response}$ を特定するには、次の数式を使用します。

$$T_{response} = n/r - T_{think}$$

次に、各引数について説明します。

- *n* は、並行ユーザーの数です。
- *r* は、サーバーが受信する 1 秒あたりの要求の数です。
- T_{think} は、平均思考時間 (秒単位) です。
  正確な応答時間結果を得るには、必ず式に思考時間を含めてください。

例 2–1 応答時間の計算

次の条件が存在する場合、

- ピーク負荷時にシステムでサポートできる並行ユーザーの最大数 (*n*) は 5,000。
- ピーク負荷時にシステムで処理できる要求の最大数 (*r*) は 1 秒あたり 1,000。

平均思考時間 (T_{think}) は 1 要求あたり 3 秒。

応答時間の計算は次のようになります。

$$T_{response} = n/r - T_{think} = (5000/1000) - 3\text{ 秒} = 5 - 3\text{ 秒}$$

したがって、応答時間は 2 秒です。

システムの (特に、ピーク負荷時の) 応答時間を計算したら、それを、アプリケーションで許容可能な応答時間と比較します。応答時間は、スループットとともに、Application Server のパフォーマンスにとって重要な主要要因の 1 つです。

1 分あたりの要求数

任意の時点での並行ユーザーの数、その要求の応答時間、およびユーザーの平均思考時間がわかっている場合は、1 分あたりの要求数を計算できます。一般には、システム上に存在する並行ユーザーの数の見積もりから始めます。

たとえば、管理者が Web サイトのパフォーマンスソフトウェアを実行したあと、オンラインバンキング Web サイトで要求を送信している並行ユーザーの平均数は 3,000 であるとの結論を出したとします。この数字は、オンライン銀行のメンバーになるためにサインアップしたユーザーの数、それらのユーザーのバンキングトランザクション行動、それらのユーザーが要求の送信を選択した日または週の時間帯などに依存します。

したがって、これらの情報がわかれば、この節で説明した 1 分あたりの要求数の数式を使用して、このユーザーベースに対してシステムが処理できる 1 分あたりの要求数を計算できます。ピーク負荷時には 1 分あたりの要求数と応答時間が反比例関係になるため、より優れた応答時間を得るためのトレードオフとして 1 分あたりの要求数を減らすことが許容可能かどうか、あるいは、1 分あたりの要求数を増やすためのトレードオフとして応答時間を遅くすることが許容可能かどうかを判断してください。

システム性能を微調整するための開始点として許容可能な 1 分あたりの要求数と応答時間のしきい値で試してください。そのあと、システムのどの領域に調整が必要かを判断してください。

前の節の式で *r* を求めると、次のようになります。

$r = n/(T_{response} + T_{think})$

例 2–2　1 秒あたりの要求数の計算

次の値の場合、

- $n = 2{,}800$ の並行ユーザー
- $T_{response} = 1$ (1 要求あたりの平均応答時間は 1 秒)
- $T_{think} = 3$ (平均思考時間は 3 秒)

1 秒あたりの要求の数の計算は次のようになります。

```
r = 2800 / (1+3) = 700
```

したがって、1 秒あたりの要求の数は 700、1 分あたりの要求の数は 42000 です。

HADB への負荷の見積もり

HADB への負荷を計算するには、次の要因を考慮します。

- 52 ページの「HTTP セッション持続性の頻度」
- 52 ページの「HTTP セッションサイズと範囲」
- 53 ページの「ステートフルセッション Bean のチェックポイント設定」

セッション持続性を設定する手順については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 9 章「高可用性 (HA) セッション持続性とフェイルオーバーの設定」を参照してください。

HTTP セッション持続性の頻度

HADB で受信される 1 分あたりの要求の数は、持続性の頻度に依存します。持続性の頻度によって、Application Server が HTTP セッションデータを HADB に保存する頻度が決定されます。

持続性の頻度オプションを次に示します。

- **web-method** (デフォルト): サーバーは、すべての HTTP 応答でセッションデータを格納します。このオプションによって、格納されたセッション情報が最新になることは保証されますが、HADB に対するトラフィックが増えます。
- **time-based**: セッションは、指定された時間間隔で格納されます。このオプションによって、HADB に対するトラフィックは削減されますが、セッション情報が最新になることは保証されません。

次の表は、持続性の頻度オプションの長所と短所を要約しています。

表 2–1 持続性の頻度オプションの比較

持続性の頻度オプション	長所	短所
web-method	最新のセッション情報を使用できることが保証されます。	応答時間が増え、スループットが低下する可能性があります。
time-based	応答時間が短縮されるとともに、スループットが向上する可能性もあります。	アプリケーションサーバーインスタンスの障害が発生したあと、最新のセッション情報を使用できる保証は低くなります。

HTTP セッションサイズと範囲

1 要求あたりのセッションサイズは、セッション内に格納されているセッション情報の量に依存します。

ヒント – 全体的なパフォーマンスを向上させるには、セッション内の情報の量をできるだけ削減します。

持続性の範囲の設定を使用して、1 要求あたりのセッションサイズを微調整することができます。HTTP セッション持続性の範囲を、次のオプションから選択します。

- **session**: サーバーは、セッション情報を HADB に保存するたびに、セッションオブジェクト全体を直列化して保存します。
- **modified-session**: サーバーは、セッションが変更された場合、そのセッションのみを保存します。サーバーは変更を、Bean の `setAttribute()` メソッドへの呼び出しを傍受することによって検出します。このオプションでは内部オブジェクトへの直接の変更は検出されないため、このような場合は、`setAttribute()` を明示的に呼び出すように SFSB をコーディングしてください。

- **modified-attribute**: サーバーは、そのセッションが最後に格納されてからあとに変更 (挿入、更新、または削除) された属性のみを保存します。これには modified-session と同じ欠点がありますが、正しく適用すれば、HADB 書き込みスループットの要件が大幅に削減される可能性があります。

このオプションを使用するには、アプリケーションが次の処理を行う必要があります。

- セッション状態を変更するたびに、`setAttribute()` または `removeAttribute()` を呼び出す。
- 属性間で相互参照しないようにする。
- 複数の属性間、または少なくとも読み取り専用属性と変更可能な属性間でセッション状態を分散する。

次の表は、持続性の範囲オプションの長所と短所を要約しています。

表 2–2 持続性の範囲オプションの比較

持続性の範囲オプション	長所	短所
modified-session	セッション状態を変更しない要求の応答時間が向上します。	Web メソッド (一般には、`doGet()` または `doPost()`) の実行中、アプリケーションは次のセッションメソッドを呼び出す必要があります。 ■ 属性が変更された場合は、`setAttribute()` ■ 属性が削除された場合は、`removeAttribute()`
session	アプリケーションに制約はありません。	`modified-session` および `modified-attribute` オプションに比べて、スループットや応答時間が低下する可能性があります。
modified-attribute	変更されたセッション状態の割合が低い要求のスループットや応答時間が向上します。	特定の要求に対する変更されたセッション状態の割合が 60% に近づくと、スループットや応答時間は低下します。このような場合は、属性を個別のレコードに分割するためのオーバーヘッドにより、パフォーマンスはほかのオプションより悪くなります。

ステートフルセッション Bean のチェックポイント設定

SFSB セッション持続性の場合、HADB への負荷は次の要素に依存します。

- チェックポイント設定が有効になっている SFSB の数。
- チェックポイント設定のために選択されている SFSB メソッドと、その使用頻度。
- セッションオブジェクトのサイズ。
- トランザクションで使用するメソッド。

チェックポイント設定は一般に、その SFSB を含む任意のトランザクションが完了したあとに (そのトランザクションがロールバックした場合でも) 発生します。

パフォーマンスを向上させるには、チェックポイント設定に指定するメソッドのセットを小さくします。チェックポイントを設定するデータのサイズと、チェックポイント設定の頻度によって、特定のクライアント対話の応答時間で発生する追加のオーバーヘッドが決定されます。

ネットワーク構成の計画

Application Server をネットワークに統合する方法を計画する場合は、帯域幅の要件を見積もり、ユーザーのパフォーマンス要件を満たすような方法でネットワークを計画します。

ここで説明する内容は次のとおりです。

帯域幅の要件の見積もり

ネットワークの望ましいサイズと帯域幅を決定するには、まずネットワークトラフィックを特定し、そのピークを識別します。全体の量がピークになる特定の時間、曜日、または月の特定の日が存在するかどうかを確認したあと、そのピークの所要時間を特定します。

ピーク負荷の時間帯に、ネットワーク内のパケットの数はもっとも高いレベルに達します。一般に、ピーク負荷の設計を行う場合は、ピーク量の 100 パーセントを処理するという目標でシステムを拡張させます。ただし、どのようなネットワークでも予期しない動作をすること、またどれだけ拡張したとしても、必ずしもピーク量の 100 パーセントを処理できるとは限らないことを念頭においてください。

たとえば、ピーク負荷時に、ユーザーの 5 パーセントが Application Server 上に配備されているアプリケーションへのアクセス中にネットワークに直接アクセスできない場合があるとします。その 5 パーセントのうち、最初の試行のあとにアクセスを再試行するユーザーがどれだけいるかを見積もってください。ここでも、それらのユーザーがすべて実行するとは限らず、その失敗したユーザーのうちの一定の割合が再試行します。その結果、ユーザーがアクセスを試行し続けるに伴って徐々にピーク使用が拡大するため、ピークが現れる時間は長くなります。

必要な帯域幅の計算

45 ページの「パフォーマンス目標の確立」で行った計算に基づいて、サイトに Application Server を配備するために必要な追加の帯域幅を決定します。

アクセス方法 (T-1 回線、ADSL、ケーブルモデム、その他) に応じて、見積もった負荷を処理するために必要な増加した帯域幅の量を計算します。たとえば、サイトで T-1 回線または高速な T-3 回線を使用しているとします。回線の帯域幅がわかったら、サイトで 1 秒あたりに生成される要求の平均数および最大ピーク負荷に基づいて、ネットワークに必要な回線数を見積もります。Web サイトの分析および監視ツールを使用して、これらの数値を計算します。

例 2–3　必要な帯域幅の計算

1 本の T-1 回線は、1.544 Mbps を処理できます。したがって、T-1 回線 4 本から成るネットワークは約 6 Mbps のデータを処理できます。クライアントに送り返される平均的な HTML ページが 30K バイトであると仮定すると、T-1 回線 4 本から成るこのネットワークは 1 秒あたり次のトラフィックを処理できます。

6,176,000 ビット/8 ビット = 1 秒あたり 772,000 バイト

1 秒あたり 772,000 バイト/30K バイト = 1 秒あたり約 25 の並行応答ページ

1 秒あたり 25 ページのトラフィックとすると、このシステムは 1 時間あたり 90,000 ページ (25 × 60 秒× 60 分) を処理できるため、負荷が 1 日中均一であると仮定した場合、1 日あたり最大 2,160,000 ページになります。最大ピーク負荷がこれより高い場合は、それに応じて帯域幅を増やします。

ピーク負荷の見積もり

負荷を 1 日中均一にすることは、おそらく現実的ではありません。ピーク負荷がいつ発生し、どれだけ持続するか、および負荷全体の何パーセントがピーク負荷になるかを特定する必要があります。

例 2–4　ピーク負荷の計算

ピーク負荷が 2 時間持続し、2,160,000 ページの負荷全体の 30 パーセントに相当する場合は、1 日のうちの 2 時間の間に T-1 回線上で 648,000 ページを処理する必要があることを示します。

したがって、この 2 時間の間にピーク負荷に対応するには、T-1 回線の数を次の計算に従って増やします。

648,000 ページ/120 分 = 1 分あたり 5,400 ページ

1 分あたり 5,400 ページ/60 秒 = 1 秒あたり 90 ページ

例 2–4 ピーク負荷の計算 (続き)

4 回線で 1 秒あたり 25 ページを処理できるとすると、その約 4 倍のページにはその 4 倍の回線 (この場合は 16 回線) が必要になります。この 16 回線は、30 パーセントのピーク負荷という現実的な最大量の処理を考慮したものです。明らかに、これらの多数の回線を使用すれば、その他の 70 パーセントの負荷を 1 日の残り時間にわたって処理できます。

サブネットの設定

アプリケーションサーバーインスタンスと HADB ノードが個別のホストマシン上に存在する分離層トポロジを使用すると、すべての HADB ノードを個別のサブネット上に配置することによってパフォーマンスを向上させることができます。これは、HADB がユーザーデータグラムプロトコル (UDP) を使用しているためです。個別のサブネットを使用すると、そのサブネットの外側にあるマシン上の UDP トラフィックが削減されます。ただし、すべての HADB ノードが同じサブネット上に存在する必要があることに注意してください。

すべてのノードと管理エージェントが同じサブネット上に存在するかぎり、管理クライアントを別のサブネットから引き続き実行できます。すべてのホストとポートをすべてのノードエージェント内でアクセス可能にするとともに、ノードをファイアウォール、UDP のブロックなどでブロックしないでください。

HADB は UDP マルチキャストを使用するため、HADB ノードを含むサブネットはすべてマルチキャスト用に設定してください。

ネットワークカードの選択

帯域幅を増やし、ネットワークパフォーマンスを最適化するために、Application Server をホストしているサーバーと HADB ノードの間に少なくとも 100 Mbps の Ethernet カード、できれば 1 Gbps の Ethernet カードを使用してください。

HADB のためのネットワーク設定

注 – HADB は UDP マルチキャストを使用するため、システムのルーターとホストのネットワークインタフェースカード上でマルチキャストを有効にしてください。HADB が複数のサブネットワークにまたがっている場合は、それらのサブネットワーク間のルーター上でもマルチキャストを有効にしてください。最適な結果を得るには、HADB ノードをすべて同じネットワーク上に配置します。アプリケーションサーバーインスタンスは、異なるサブネットワーク上に配置できます。

次の提案を採用すると、HADB をネットワーク内で最適な状態で動作させることができます。

- 各ネットワークインタフェースに 100 Mbps 以上の専用の Ethernet チャネルが割り当てられるように、スイッチ型のルーターを使用します。
- 4 つ以上の HADB ノードをホストしているマルチ CPU マシン上で HADB を実行する場合は、1 Gbps の Ethernet カードを使用します。平均セッションサイズが 50K バイトより大きい場合は、マシンあたりの HADB ノード数が 4 未満の場合でも、1 Gbps の Ethernet カードを使用します。
- HADB ノード内にネットワークボトルネックの疑いがある場合は、次のようにします。
  - HADB サーバー上でネットワーク監視ソフトウェアを実行して、問題を診断します。
  - ネットワーク内の 100 Mbps の Ethernet カードをすべて 1 Gbps の Ethernet カードに置き換えることを検討します。

可用性のための計画

ここでは、次の内容について説明します。

可用性の規模の適正化

システムやアプリケーションの可用性を計画するには、異なるアプリケーションにアクセスするユーザーグループの可用性ニーズを評価します。たとえば、料金を支払う外部のユーザーやビジネスパートナーはたいてい、サービスの品質 (QoS) に内部のユーザーより高い期待を持っています。そのため、アプリケーション機能、アプリケーション、サーバーなどが使用不可になっても、支払いを行う外部の顧客に比べて内部のユーザーの方が許容性が高い可能性があります。

次の図は、発生確率が低い出来事ほど、影響を軽減するためのコストと複雑さが増加するようすを示しています。この連続曲線の 1 端では、単純な負荷分散クラスタを使用して、局所的なアプリケーション、ミドルウェア、およびハードウェア障害に耐えることができます。曲線のもう一方の端では、地理的に孤立したクラスタを使用することにより、データセンター全体に影響する大災害を軽減できます。

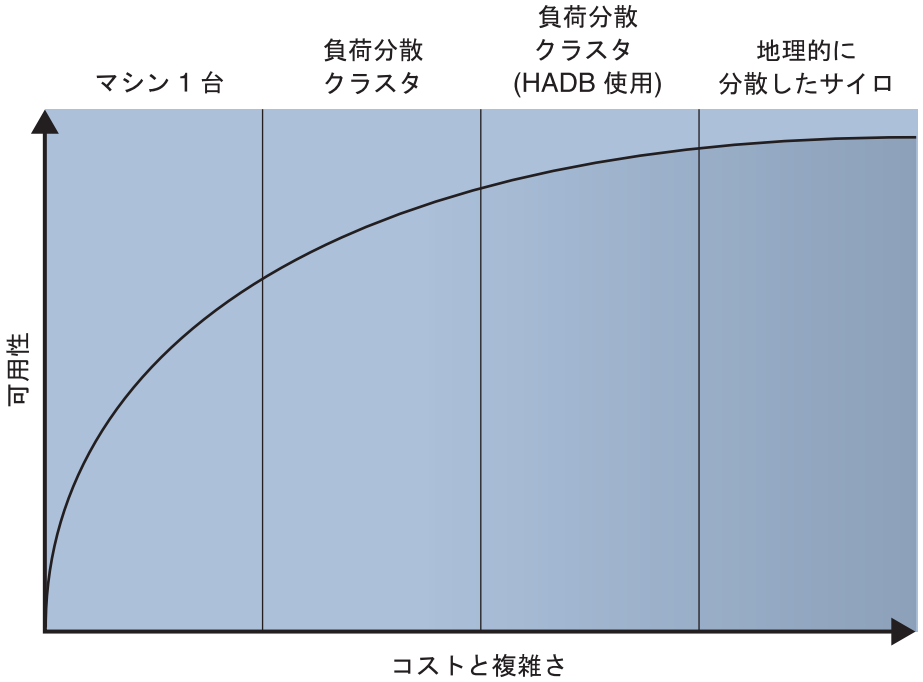


図 2–3　可用性に対応したコストと複雑さ

高い投資収益率を実現するには、多くの場合、アプリケーション内の機能の可用性の要件を特定することが有効です。たとえば、保険見積もりシステムが使用不可になることは許容されない (それによって新しい企業を失う) 可能性がありますが、既存の顧客が現在の対象範囲を表示できるアカウント管理機能が短期間使用不可になっても、既存の顧客を失うことはほとんどありません。

可用性を向上させるためのクラスタの使用

もっとも基本的なレベルで言えば、クラスタとは、クライアントには 1 つのインスタンスとして見えるアプリケーションサーバーインスタンスのグループであり、たいていは複数の物理サーバー上にホストされています。これにより、水平方向のスケーラビリティーや、1 台のマシン上の 1 つのインスタンスより高い可用性が提供されます。この基本的なレベルのクラスタ分布が Application Server の HTTP ロードバランサプラグインと連携して動作します。このプラグインは、HTTP および HTTPS 要

求を受け付け、それをクラスタ内のアプリケーションサーバーインスタンスの 1 つに転送します。また、ORB や、統合された JMS ブローカも、アプリケーションサーバークラスタへの負荷分散を実行します。ネットワーク障害のためにインスタンスが失敗して使用不可になるか、または応答しなくなると、要求は既存の使用可能なマシンにのみリダイレクトされます。ロードバランサはまた、失敗したインスタンスが復旧したことを認識し、それに応じて負荷を再配分することもできます。

HTTP ロードバランサにはまた、サーバーや特定の URL を監視してそれらが使用可能かどうかを判定できる診断プログラムも用意されています。診断プログラムのオーバーヘッドを、それ自体が処理の大きな負荷にならないように慎重に管理してください。

状態を持たないアプリケーションや、低い値の単純なユーザートランザクションのみが含まれるアプリケーションには、たいてい、負荷分散された単純なクラスタがあれば十分です。状態のあるミッションクリティカルなアプリケーションに対しては、セッション持続性のために HADB を使用することを考慮してください。HADB の概要については、『Application Server 管理ガイド』の第 1 章にある 34 ページの「高可用性データベース」を参照してください。

アプリケーションのオンラインアップグレードを実行するには、アプリケーションサーバーインスタンスを複数のクラスタにグループ化する方法が最適です。Application Server には、アプリケーションとインスタンスの両方を休止する機能があります。休止とは、インスタンス (またはインスタンスのグループ) あるいは特定のアプリケーションを、現在そのインスタンスまたはアプリケーションからサービスを受けているユーザーに影響を与えることなく、制御された方法でオフラインにする機能です。あるインスタンスが休止されると、新しいユーザーは、別のインスタンス上のアップグレードされたアプリケーションからサービスを受けます。この種類のアプリケーションアップグレードは、順次アップグレードと呼ばれます。生存中のアプリケーションのアップグレードの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「可用性を低下させないアプリケーションのアップグレード」を参照してください。

システムへの冗長性の追加

高可用性を実現するための 1 つの方法は、システムにハードウェアやソフトウェアの冗長性を追加することです。あるユニットに障害が発生すると、冗長なユニットが引き継ぎます。これは、耐障害性とも呼ばれます。一般に、高可用性を最大化するには、システム内に存在する可能性のあるすべての単点障害を特定して削除します。

障害クラスの識別

冗長性のレベルは、システムで許容する必要のある障害クラス (障害の種類) によって決定されます。障害クラスのいくつかの例を次に示します。

- システムプロセス

- マシン
- 電源装置
- ディスク
- ネットワーク障害
- ビル火災またはその他の予防可能な災害
- 予測できない天災

重複したシステムプロセスによって、単一のシステムプロセス障害や単一のマシン障害に耐えることができます。重複した、ミラー化された (ペアになった) マシンを異なる電源装置に接続することにより、単一の停電に耐えることができます。ミラー化されたマシンを個別のビル内に保持することにより、単一のビル火災に耐えることができます。ミラー化されたマシンを地理的に離れた場所に保持することにより、地震などの天災に耐えることができます。

可用性を向上させるための HADB 冗長ユニットの使用

45 ページの「パフォーマンス目標の確立」で説明したように、可用性を向上させるために、HADB ノードは常にデータ冗長ユニット (DRU) で使用されます。

耐障害性を向上させるための HADB スペアノードの使用

スペアノードを使用すると、耐障害性が向上します。スペアノードは必須ではありませんが、最大限の可用性を実現します。

フェイルオーバー容量の計画

フェイルオーバー容量の計画では、サーバーまたはプロセスに障害が発生した場合にシステムがデータをシームレスに復元して処理を続行できるようにするには Application Server 配備にサーバーやプロセスをどれだけ追加する必要があるかを決定します。システムが過負荷になると、プロセスまたはサーバー障害が発生して、応答時間の低下や、場合によってはサービスの完全な停止を引き起す可能性があります。このような状況に対して準備することは、配備を成功させるために重要です。

容量 (特に、ピーク負荷時の容量) を維持するには、既存の配備に、Application Server インスタンスを実行しているスペアマシンを追加します。

たとえば、それぞれ 1 つの Application Server インスタンスを実行している 2 台のマシンから成るシステムを考えてみます。これらのマシンが合わせて、1 秒あたり 300 要求のピーク負荷を処理しています。これらのマシンの 1 つが使用不可になった場合は、マシン間の負荷分散が均一であると仮定すると、システムは 150 の要求しか処理できません。したがって、ピーク負荷時の要求の半分はサービスを受けられなくなります。

設計上の決定

設計上の決定には、ピーク負荷または通常状態負荷のどちらのためにシステムを設計するか、および各種のロールにあるマシンの数とそれらのサイズが含まれます。

ピーク負荷または通常状態負荷のための設計

標準的な配備では、通常状態とピークの作業負荷の間に違いが存在します。

- システムがピーク負荷を処理するように設計されている場合は、応答時間を低下させることなく、ユーザーや要求の予測される最大の負荷に耐えることができます。これは、システムが、予測されるシステム負荷の極端な場合を処理できることを示します。ピーク負荷と通常状態負荷の違いが大きい場合は、ピーク負荷のために設計すると、アイドル状態の場合が多いリソースにお金をかけることになります。
- システムが通常状態負荷を処理するように設計されている場合は、予測されるピーク負荷を処理するために必要なリソースの一部が存在しません。そのため、ピーク負荷が発生すると、システムの応答時間が低下します。

システムでピーク負荷をどれだけ頻繁に処理すると予測されるかによって、ピーク負荷または通常状態負荷のどちらのために設計するかが決定されます。

ピーク負荷が頻繁に、たとえば 1 日に数回発生する場合は、それを処理できるように容量を拡張するだけの価値があるかもしれません。システムが時間全体の 90 パーセントを通常状態で動作し、ピーク負荷での動作が 10 パーセントしかない場合は、通常状態負荷のために設計されたシステムを配備する方が望ましい可能性があります。これは、システムの応答時間が時間全体の 10 パーセントだけ低下することを示します。システムがピーク負荷で動作する頻度または所要時間によって、システムにリソースを追加する必要性が正当化されるかどうかを判断してください。

システムのサイジング

アプリケーションサーバーインスタンスへの負荷、HADB への負荷、およびフェイルオーバーの要件に基づいて、次の値を決定できます。

- 62 ページの「アプリケーションサーバーインスタンスの数」
- 62 ページの「HADB ノードの数」
- 63 ページの「HADB ホストの数」
- 63 ページの「HADB のストレージ容量」

アプリケーションサーバーインスタンスの数

必要なアプリケーションサーバーインスタンス (ホスト) の数を決定するには、各インスタンスが複数の中央処理装置 (CPU) を使用できるとしても、47 ページの「Application Server インスタンスへの負荷の見積もり」で説明した各アプリケーションサーバーインスタンスに対する要因に基づいて環境を評価します。

HADB ノードの数

一般的なガイドラインとして、システム内の各 CPU に 1 つの HADB ノードを割り当てるように計画します。たとえば、2 つの CPU を備えたマシンには 2 つの HADB ノードを使用します。

注 – マシンあたりに複数の HADB ノードを割り当てている (たとえば、大型のマシンを使用している) 場合は、マシン上に十分な冗長性とスケーラビリティー (たとえば、複数の無停電電源装置や独立したディスク制御装置) が必ず存在するようにしてください。

あるいは、次の手順を使用します。

▼ 必要な HADB ノードの数を決定するには

1 次のパラメータを決定します。

- 並行ユーザーの最大数、n_{users}。
- 平均の BLOB サイズ、s。
- NTPS と呼ばれる、ユーザーあたりの最大トランザクション比率。

2 最大プライマリデータ量の G バイトのサイズ、V_{data} を決定します。

次の数式を使用します。

$$V_{data} = n_{users} \cdot s$$

3 最大 HADB データ転送速度、R_{dt} を決定します。

この値には、アプリケーション側から HADB に転送されるデータ量が反映されます。次の数式を使用します。

$$R_{dt} = n_{users} \cdot s \cdot NTPS$$

4 ノードの数、N_{NODES} を決定します。

次の数式を使用します。

$$N_{NODES} = V_{data} / 5GB$$

ノードはペアで動作するため、この値を偶数に丸めます。

HADB ホストの数

データ転送の要件に基づいて、HADB ホストの数を決定します。この計算では、すべてのホストが同じハードウェア構成とオペレーティングシステムを備え、かつ実行するノードを格納するために必要なリソースを含んでいることを前提にしています。

▼ ホストの数を計算するには

1 最大ホストデータ転送速度、R_{max} を決定します。

この値はネットワークやホストのハードウェアによって異なるため、経験に基づいて決定します。この値は、前の節で決定した最大 HADB データ転送速度、R_{dt} とは異なることに注意してください。

2 このデータを格納するために必要なホストの数を決定します。

ホストの数 N_{HOSTS} に分散されるデータ量 V を更新すると、各ホストは約 $4V/N_{HOSTS}$ のデータを受信するようになります。次の数式を使用して、このデータ量を格納するために必要なホストの数を決定します。

$$N_{HOSTS} = 4 \cdot R_{dt} / R_{max}$$

各 DRU に対するホストの数を同じにするために、この値をもっとも近い偶数に丸めます。

3 スペアノードとして、各 **DRU** に **1** 台のホストを追加します。

ほかの各ホストが N データノードを実行する場合は、このホストで N スペアノードを実行するようにします。これにより、単一のマシン障害で N データノードがダウンするようになります。

各ホストは少なくとも 1 つのノードを実行する必要があるため、ノードの数がホストの数より小さい場合 ($N_{NODES} < N_{HOSTS}$) は、N_{NODES} を N_{HOSTS} に等しくなるように調整します。ノードの数がホストの数より大きい場合 ($N_{NODES} \backslash > N_{HOSTS}$) は、同じホスト上でいくつかのノードを実行できます。

HADB のストレージ容量

HADB では、ネットワークの容量を超えるまで、ノードの追加によってほぼリニアなスケーリングが得られます。各ノードでは、専用のディスク (1 台または複数台) 上にストレージデバイスが設定されている必要があります。すべてのノードで、ストレージデバイスに等しい容量が割り当てられている必要があります。ストレージデバイスは必ずローカルディスクに割り当ててください。

予測されるセッションデータサイズが *x* M バイトであるとします。ここで、*x* M バイトは、システム全体のストレージの合計容量です (つまり、*x* = N ユーザー * s)。HADB はミラーノード上のデータをレプリケートするため、2*x* M バイトのストレージが必要です。さらに、HADB は、データへの高速アクセスを可能にするためにイ

ンデックスを使用しています。2 つのノードにはインデックス用に追加の 2*x* M バイトが必要であり、必要な合計ストレージ容量は 4*x* なります。したがって、HADB の予測されるストレージ容量の要件は、予測されるデータ量の 4 倍です。

新しいノードを追加したあとにデータの再断片化が必要になる可能性があるため、HADB からデータが失われることなく将来も拡張できるようにするには、オンラインアップグレードのための追加のストレージ容量を用意してください。この場合は、データデバイス上に同じ量 (4*x*) の追加の領域が必要です。そのため、予測されるストレージ容量は、予測されるデータ量の 8 倍になります。

さらに、HADB はディスク容量を次のように使用します。

- ログバッファーの一時記憶域のための領域。この領域は、ログバッファーサイズの 4 倍です。ログバッファーは、データに関連する操作を常時監視します。ログバッファーサイズのデフォルト値は 48M バイトです。
- 内部の管理のための領域。この領域は、ストレージデバイスサイズの 1 パーセントです。

次の表は、*x* M バイトの合計セッションデータサイズのための HADB ストレージ領域の要件を要約しています。この値は、HADB データベースのすべてのノードにわたって分散したセッションストレージデータの合計容量であることに注意してください。ノードあたりのデバイスサイズは、そのノード用に指定されたすべてのデバイスで共有されます。

表 2–3 ×M バイトの合計セッションデータサイズのための HADB ストレージ領域の要件

条件	必要な HADB ストレージ領域
オンラインが必要でない場合の HADB ノードの追加または削除。	(4*x*/N ノード) M バイト + (4 ×ログバッファーサイズ) + デバイスサイズの 1%
オンラインが必要な場合の HADB ノードの追加または削除。	(8*x*/N ノード) M バイト + (4 ×ログバッファーサイズ) + デバイスサイズの 1%

HADB のデバイス領域が不足すると、HADB は、データを挿入または更新するクライアント要求を受け付けません。ただし、削除操作は受け付けます。HADB のデバイス領域が不足した場合は、エラーコード 4593 または 4592 を返し、対応するエラーメッセージを履歴ファイルに書き込みます。これらのメッセージの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Error Message Reference』の第 14 章「HADB Error Messages」を参照してください。

Message Queue ブローカの配備の計画

Java Message Service (JMS) API は、J2EE アプリケーションおよびコンポーネントに対して、メッセージの作成、送信、受信、および読み取りを可能にするメッセージング標準です。この API によって、緩やかに結合され、信頼性が高く、非同期の分散通信が可能となります。Sun Java System Message Queue 3 は JMS を実装し、Application Server と統合されているため、MQ を使用してメッセージ駆動型 Bean (MDB) などのコンポーネントを作成できます。

Sun Java System Message Queue (MQ) は、J2EE コネクタアーキテクチャー仕様 (JCA) 1.5 で規定されているように、コネクタモジュール (リソースアダプタともいう) を使用して Application Server に統合されています。コネクタモジュールは、Application Server に機能を追加するための標準化された方法です。Application Server に配備された J2EE コンポーネントは、コネクタモジュールによって統合された JMS プロバイダを使用して JMS メッセージを交換します。この JMS プロバイダは、デフォルトでは Sun Java System Message Queue ですが、JCA 1.5 が実装されているかぎり、必要に応じて別の JMS プロバイダを使用できます。

Application Server で JMS リソースを作成すると、バックグラウンドでコネクタリソースが作成されます。そのようにして、JMS 操作のたびにコネクタランタイムが呼び出され、バックグラウンドで MQ リソースアダプタが使用されます。

リソースアダプタ API の使用に加えて、Application Server は、MQ とのより緊密な統合を実現するために MQ API を使用します。この緊密な統合によって、コネクタのフェイルオーバー、アウトバウンド接続の負荷分散、MDB へのインバウンドメッセージの負荷分散などの機能が可能になります。これらの機能を使用すると、メッセージングトラフィックの耐障害性や高可用性を実現できます。

マルチブローカクラスタ

MQ Enterprise Edition は、ブローカクラスタと呼ばれる、複数の相互接続されたブローカインスタンスの使用をサポートしています。ブローカクラスタによって、クライアント接続はクラスタ内のすべてのブローカに分散されます。クラスタ化することで、水平方向のスケーラビリティーが提供され、可用性が向上します。

1 つのメッセージブローカが約 8 個の CPU まで拡張して、標準的なアプリケーションのための十分なスループットを提供します。ブローカプロセスが失敗した場合、そのプロセスは自動的に再開されます。ただし、ブローカーに接続するクライアントの数や配信されるメッセージの数が増えると、ブローカーは最終的に、ファイル記述子の数やメモリーなどの制限を超えてしまいます。

クラスタ内に 1 つのブローカではなく複数のブローカを配置すると、次のことが可能になります。

- 1 台のマシンでハードウェア障害が発生した場合でもメッセージングサービスを提供する。

- システム保守を実行している間の停止時間を最小限に抑える。
- 異なるユーザーリポジトリを持つワークグループを格納する。
- ファイアウォールの制限に対応する。

ただし、複数のブローカを配置しても、ブローカ障害の時点で進行中であったトランザクションが代替ブローカに引き継がれることは保証されません。MQ は、失敗した接続をクラスタ内の別のブローカを使用して再確立しますが、トランザクションメッセージングを失い、進行中のトランザクションをロールバックします。完了できなかったトランザクションを除き、ユーザーアプリケーションは影響されません。接続は引き続き使用可能であるため、サービスのフェイルオーバーは保証されます。

そのため、MQ はクラスタ内の高可用性持続メッセージをサポートしていません。障害のあとブローカを再起動すると、ブローカは自動的に回復し、持続メッセージの配信を完了します。持続メッセージはデータベース内か、またはファイルシステムに格納される可能性があります。ただし、ブローカをホストしているマシンがハード障害から回復しない場合は、メッセージが失われる可能性があります。

Sun Cluster Data Service for Sun Message Queue を備えた Solaris プラットフォームは、持続メッセージの透過的なフェイルオーバーをサポートしています。この構成は、真の高可用性を実現するために Sun Cluster のグローバルファイルシステムと IP フェイルオーバーを利用しており、また Java Enterprise System にも含まれています。

マスターブローカとクライアントの同期

マルチブローカ構成では、各送信先がクラスタ内のすべてのブローカにレプリケートされます。各ブローカは、ほかのすべてのブローカ上の送信先として登録されているメッセージコンシューマを認識しています。そのため、各ブローカは、独自の直接接続されているメッセージプロデューサから遠隔メッセージコンシューマにメッセージを経路指定したり、遠隔プロデューサから独自の直接接続されているコンシューマにメッセージを配信したりすることができます。

クラスタ構成では、各メッセージプロデューサが直接接続されているブローカが、そのプロデューサから送信されるメッセージの経路指定を実行します。そのため、持続メッセージの格納と経路指定の両方を、そのメッセージのホームブローカが実行します。

管理者がブローカ上の送信先を作成または破棄した場合は常に、この情報がクラスタ内のほかのすべてのブローカに自動的に伝播されます。同様に、メッセージコンシューマがホームブローカに登録された場合、またはコンシューマがホームブローカから切り離された (明示的か、クライアントまたはネットワークの障害のためか、ホームブローカがダウンしたためかにかかわらず) 場合は常に、そのコンシューマに関連する情報がクラスタ全体にわたって伝播されます。同様の方法で、持続性サブスクリプションに関する情報もクラスタ内のすべてのブローカに伝播されます。

Message Queue ブローカを使用するための Application Server の設定

Application Server の Java Message Service は、Message Queue のコネクタモジュール (リソースアダプタ) を表します。Java Message Service は、管理コンソールまたは asadmin コマンド行ユーティリティーから管理することができます。

MQ ブローカ (JMS ホスト) は、Application Server プロセスとは別の JVM で動作します。これにより、複数の Application Server インスタンスまたはクラスタが MQ ブローカの同じセットを共有できます。

Application Server では、*JMS* ホストは MQ ブローカを指します。Application Server の Java Message Service 設定には、使用されるすべての JMS ホストを含む JMS ホストリスト (AddressList とも呼ばれる) が含まれています。

管理コンソールを使用した JMS の管理

管理コンソールでは、特定の構成の Java Message Service ノードを使用して JMS プロパティーを設定できます。「再接続間隔」や「再接続試行」などのプロパティーを設定できます。詳細については、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』の第 4 章「Java Message Service (JMS) リソースの設定」を参照してください。

「java メッセージサービス」ノードの下の「JMS ホスト」ノードには、JMS ホストのリストが含まれています。リストにホストを追加することも、リストからホストを削除することもできます。ホストごとに、ホスト名、ポート番号、および管理ユーザーの名前とパスワードを設定できます。JMS ホストリストには、デフォルトで、Application Server に統合されたローカルの MQ ブローカを表す、"default_JMS_host" という 1 つの MQ ブローカが含まれています。

JMS ホストリストを、クラスタ内のすべての MQ ブローカを含むように設定します。たとえば、3 つの MQ ブローカを含むクラスタを設定するには、それぞれに対して Java Message Service 内に 1 つの JMS ホストを追加します。Message Queue クライアントは、Java Message Service 内の設定情報を使用して MQ ブローカと通信します。

asadmin を使用した JMS の管理

管理コンソールに加えて、asadmin コマンド行ユーティリティーでも Java Message Service と JMS ホストを管理できます。次の asadmin コマンドを使用します。

- Java Message Service 属性の設定: `asadmin set`
- JMS ホストの管理:
  - `asadmin create-jms-host`
  - `asadmin delete-jms-host`
  - `asadmin list-jms-hosts`

JMS リソースの管理:

- `asadmin create-jms-resource`
- `asadmin delete-jms-resource`
- `asadmin list-jms-resources`

  これらのコマンドの詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Reference Manual 』または対応するマニュアルページを参照してください。

Java Message Service タイプ

Application Server と MQ ブローカの間の統合には、ローカルとリモートの 2 つのタイプがあります。このタイプ属性は、管理コンソールの「Java メッセージサービス」ページで設定できます。

ローカルの Java Message Service

Type 属性が LOCAL の場合は、Application Server が MQ ブローカを起動および停止します。Application Server は起動時に、デフォルト JMS ホストとして指定されている MQ ブローカを起動します。同様に、Application Server インスタンスは停止時に、MQ ブローカを停止します。LOCAL 型はスタンドアロンの Application Server インスタンスに最適です。

LOCAL 型では、「起動引数」属性を使用して MQ ブローカの起動パラメータを指定します。

リモートの Java Message Service

Type 属性が REMOTE の場合、Application Server は、外部に設定されているブローカまたはブローカクラスタを使用します。この場合、MQ ブローカの起動と停止は Application Server とは別個に行い、MQ ツールを使用してブローカまたはブローカクラスタを設定および調整する必要があります。REMOTE 型は Application Server クラスタに最適です。

REMOTE 型では、MQ ツールを使用して MQ ブローカ起動パラメータを指定する必要があります。「起動引数」属性は無視されます。

デフォルト JMS ホスト

デフォルト JMS ホストは、管理コンソールの「Java メッセージサービス」ページで指定できます。Java Message Service タイプが LOCAL の場合、Application Server は、Application Server インスタンスが起動したときにデフォルト JMS ホストを起動します。

MQ ブローカクラスタを使用するには、デフォルト JMS ホストを削除したあと、クラスタ内のすべての MQ ブローカを JMS ホストとして追加します。この場合、デフォルト JMS ホストは JMS ホストリスト内の最初の JMS ホストになります。

また、デフォルト JMS ホストを、いずれかの JMS ホストに明示的に設定することもできます。Application Server が Message Queue クラスタを使用する場合、デフォルト JMS ホストは MQ 固有のコマンドを実行します。たとえば、MQ ブローカクラスタの物理送信先が作成される場合、デフォルト JMS ホストはその物理送信先を作成するためにコマンドを実行しますが、クラスタ内のすべてのブローカがその物理送信先を使用します。

配備シナリオの例

メッセージングのニーズに対応するには、Java Message Service と JMS ホストリストを、配備、パフォーマンス、および可用性のニーズを満たすように変更します。以降の節では、いくつかの標準的なシナリオについて説明します。

メッセージングのニーズが Application Server に関してだけでない場合、最高の可用性を得るには、MQ ブローカと Application Server を別のマシンに配備します。別のオプションとして、十分なメッセージング容量が存在するようになるまで、Application Server インスタンスと MQ ブローカインスタンスを各マシンで実行する方法があります。

デフォルト配備

Application Server をインストールすると、ドメイン管理サーバー (DAS) が自動的に作成されます。デフォルトでは、DAS の Java Message Service タイプは LOCAL です。そのため、DAS を起動すると、そのデフォルト MQ ブローカも起動されます。

新しいドメインを作成すると、新しいブローカも作成されます。デフォルトでは、ドメインにスタンドアロンのサーバーインスタンスまたはクラスタを追加すると、その Java Message Service は REMOTE として設定され、デフォルト JMS ホストは DAS によって起動されるブローカになります。

次の図は、3 つのインスタンスを含む Application Server クラスタが含まれたデフォルト配備の例を示しています。

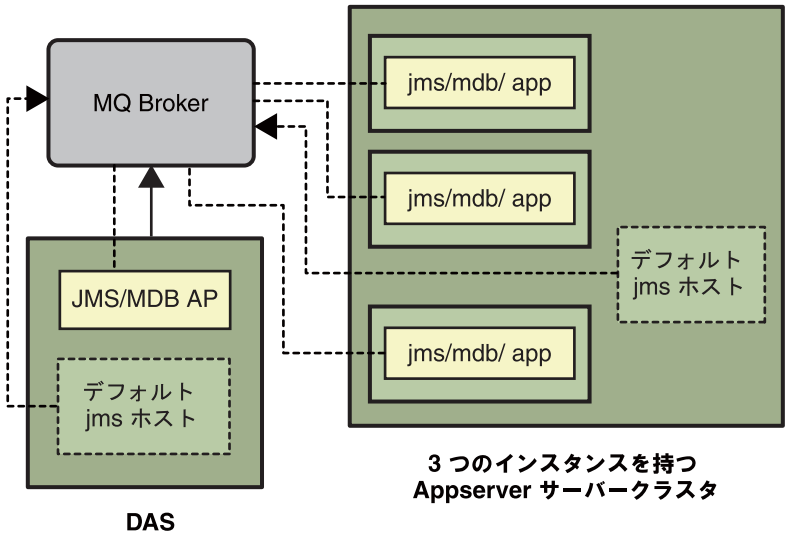


図 2–4　デフォルトの MQ 配備

Application Server クラスタでの MQ ブローカクラスタの使用

MQ ブローカクラスタを使用するように Application Server クラスタを設定するには、Application Server の Java Message Service で、すべての MQ ブローカを JMS ホストとして追加します。それにより、作成された JMS 接続ファクトリや、配備された MDB はすべて、指定された JMS 設定を使用するようになります。

次の図は、ブローカクラスタ内に 3 つの MQ ブローカと、クラスタ内に 3 つの Application Server インスタンスを含む配備の例を示しています。

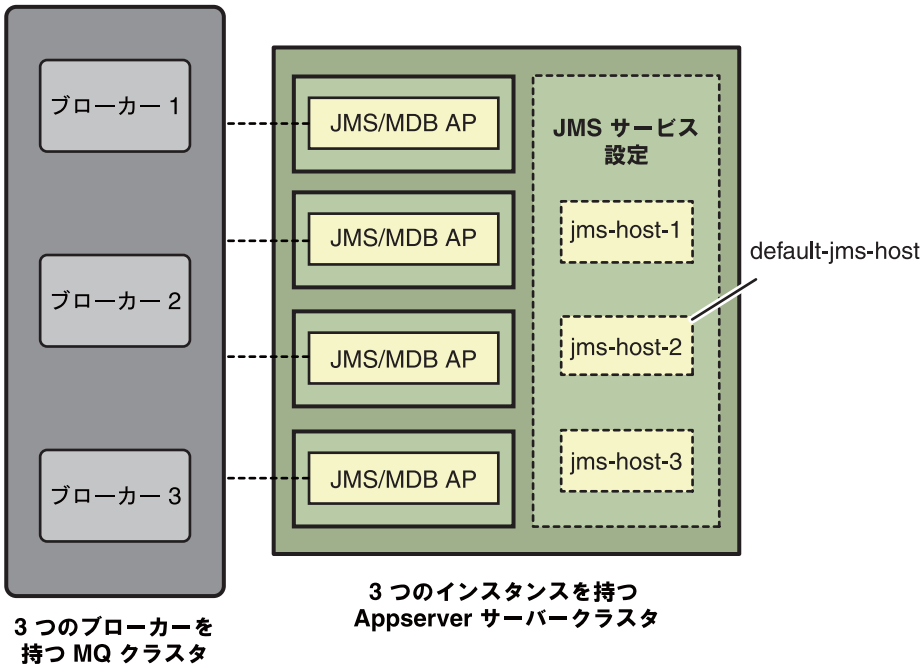


図 2–5　MQ ブローカクラスタを使用した Application Server クラスタ

アプリケーション固有の MQ ブローカクラスタの指定

場合によっては、アプリケーションが、Application Server クラスタで使用されているものとは別の MQ ブローカクラスタを使用しなければならないことがあります。次の図は、このようなシナリオの例を示しています。これを行うには、JMS 接続ファクトリの `AddressList` プロパティー、または MDB 配備記述子内の `activation-config` 要素を使用して MQ ブローカクラスタを指定します。

接続ファクトリの設定の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』の「JMS 接続ファクトリ」を参照してください。MDB の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide』の「Using Message-Driven Beans」を参照してください。

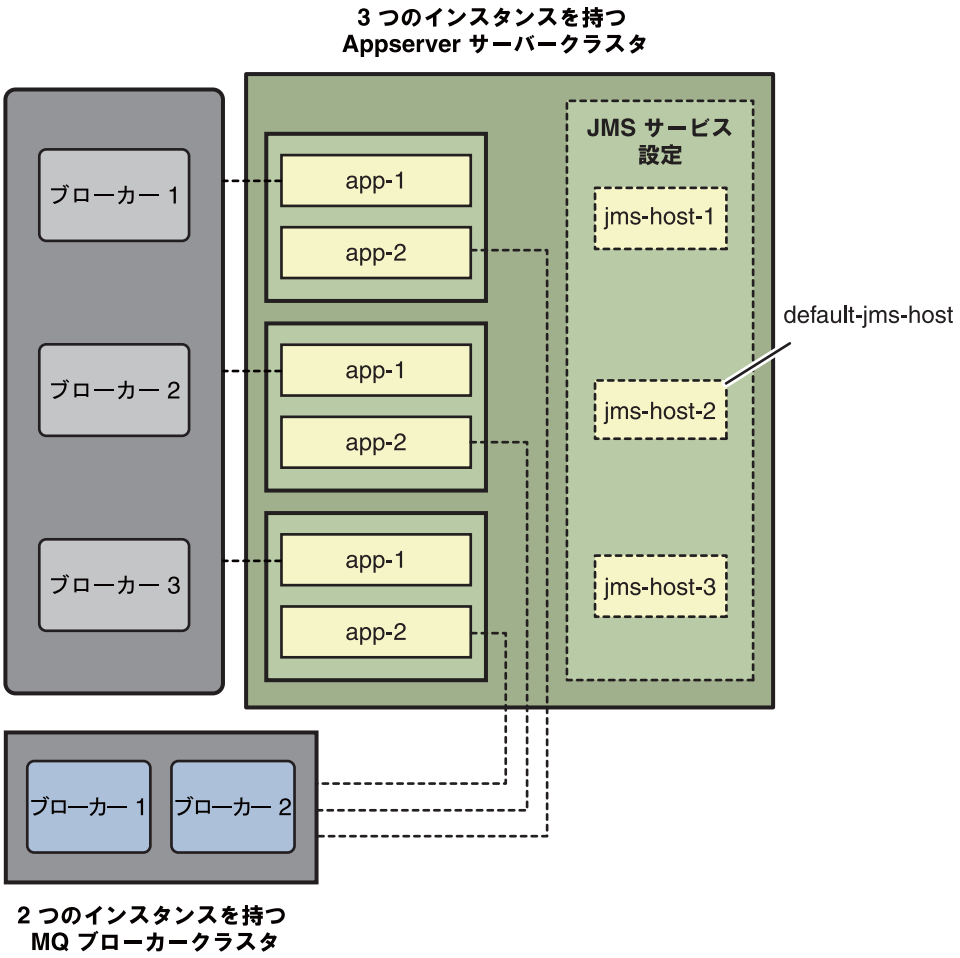


図 2–6　アプリケーション固有の MQ ブローカクラスタ

アプリケーションクライアント

アプリケーションクライアントまたはスタンドアロンのアプリケーションが、JMS 管理によるオブジェクトにはじめてアクセスすると、クライアント JVM はサーバーから Java Message Service 設定を取得します。JMS サービスへのそれ以上の変更は、JMS サービスが再開されるまで、クライアント JVM には使用できません。

◆ ◆ ◆ 第 3 章

トポロジの選択

パフォーマンスに関連した要因を 第 1 章の説明に従って見積もったあと、Application Server のトポロジを選択します。トポロジとは、マシン、Application Server インスタンス、および HADB ノードの配置や、それらの間の通信フローのことです。

2 つの基本的な配備トポロジが存在します。どちらのトポロジにも、クラスタ内の複数の Application Server インスタンス、HADB ノードのミラー化されたセット、および HADB スペアノードという共通の構成要素が含まれています。どちらのトポロジも、正しく機能するには一連の共通の設定が必要です。

この章の内容は次のとおりです。

- 両方のトポロジの 73 ページの「共通の要件」。
- 次の 2 つのトポロジ:
  - 75 ページの「共存トポロジ」 - Application Server インスタンスと HADB ノードが同じマシン上に存在する。
  - 79 ページの「分離層トポロジ」 - Application Server インスタンスと HADB ノードが別のマシン上に存在する。
- 83 ページの「使用するトポロジの決定」

共通の要件

ここでは、両方のトポロジに共通の要件について説明します。

- 73 ページの「基本要件」
- 74 ページの「HADB ノードとマシン」
- 75 ページの「ロードバランサの設定」

基本要件

どちらのトポロジも、次の基本要件を満たす必要があります。

- HADB ノードをホストするマシンはペアになっている必要がある。つまり、これらのマシンの台数は偶数である必要があります。
- 各データ冗長ユニット (DRU) 内のマシンの台数は同じである必要がある。ミラー化された (ペアになった) ノードが、主ノードとは別の DRU に含まれるように HADB データベースを作成します。
- HADB ノードをホストする各マシンにローカルのディスク記憶装置があり、その装置を使用して HADB 内のすべての持続情報を格納する必要がある。
- HADB ノードをホストするマシンは、同じオペレーティングシステムを実行している必要がある。構成とパフォーマンスの点から見て、同一またはほぼ同一のマシンを使用すると最適です。
- HTTP セッション情報および SFSB セッション情報を HADB まで持続させるには、Application Server インスタンスはクラスタ内にあり、関連するすべての要件を満たしている必要がある。クラスタの設定の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 6 章「Application Server クラスタの使用」を参照してください。
- Application Server インスタンスをホストしているマシンは、構成とパフォーマンスの点から見て、できるだけ同一である必要がある。これは、ロードバランサプラグインが負荷分散にラウンドロビン方式を使用しており、異なるクラスのマシンがインスタンスをホストしている場合は、これらのマシン全体にわたって負荷が最適には分散されないからです。
- できれば、DRU ごとに個別の無停電電源装置 (UPS) を用意する。

HADB ノードとマシン

各 DRU には HADB 内のデータの完全なコピーが含まれているため、ほかの DRU が使用不可になっても引き続き要求に対応できます。ただし、ある DRU 内のノードと、別の DRU にあるそのノードのミラーに同時に障害が発生した場合は、データの一部が失われます。そのため、停電やディスク障害など、単一の障害によって両方の DRU が影響を受けることがないようにシステムを設定することが重要です。

注 – 各 DRU を、完全に独立した冗長システム上で実行してください。

HADB ノードとマシンを設定する場合は、次のガイドラインに従ってください。

- 容量とスループットを向上させるには、ノードを各 DRU に対して 1 つずつ、ペアで追加します。
- 各マシン上で動作しているノードの数と同数のスペアノードを持たせて各 DRU を設定します。これは、構成内の各マシンが *n* 個のデータノードを実行している場合は、1 台のマシンの障害によって *n* 個のノードがダウンするためです。

- 負荷をできるだけ均等に分散するために、すべてのマシン上で同じ数の HADB ノードを実行します。

注意 – 同じマシン上の異なる DRU からノードを実行しないでください。同じマシン上の異なる DRU からノードを実行する必要がある場合は、そのマシンが任意のシングルポイント障害 (ディスク、メモリー、CPU、電源、オペレーティングシステムのクラッシュなどに関連した障害) を必ず処理できるようにしてください。

ロードバランサの設定

いずれのトポロジも、Application Server インスタンスはクラスタ内にあります。これらのインスタンスは、セッション情報を HADB まで持続させます。クラスタ内のすべての Application Server インスタンスの設定情報を含むようにロードバランサを設定してください。

クラスタの設定、およびクラスタへの Application Server インスタンスの追加の詳細については、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 6 章「Application Server クラスタの使用」を参照してください。

共存トポロジ

共存トポロジでは、Application Server インスタンスと HADB ノードが同じマシン上に存在します。このため、共存トポロジと呼ばれています。このトポロジに必要なマシンの数は、分離層トポロジに比べて少なくなります。共存トポロジでは、CPU がより効率的に使用されます。つまり、Application Server インスタンスと HADB ノードが 1 台のマシンを共有するため、処理がそれらの間で均等に分散されます。

このトポロジには、少なくとも 2 台のマシンが必要です。スループットを向上させるには、より多くのマシンをペアで追加します。

注 – 共存トポロジは、マシンの処理能力をフルに活用できるため、大規模な対称多重処理 (SMP) マシンに適しています。

構成例

次の図は、共存トポロジの構成例を示しています。

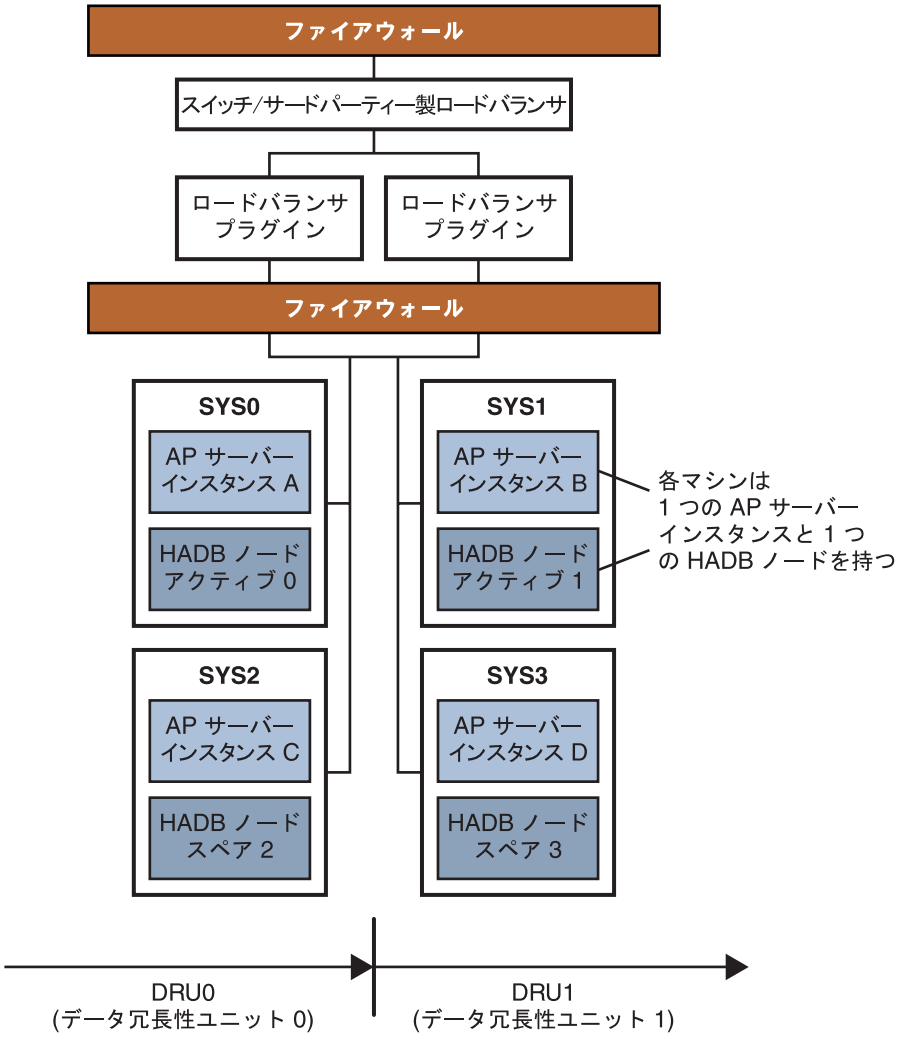


図 3–1 共存トポロジの例

マシン SYS0 は Application Server インスタンス A を、マシン SYS1 は Application Server インスタンス B を、マシン SYS2 は Application Server インスタンス C を、マシン SYS3 は Application Server インスタンス D をそれぞれホストしています。

これらの 4 つのインスタンスは、情報を次の 2 つの DRU に持続させるクラスタを形成します。

- **DRU0** は 2 台のマシン、SYS0 と SYS2 で構成されます。HADB ノードのアクティブ 0 はマシン SYS0 上にあります。HADB ノードのスペア 2 はマシン SYS2 上にあります。

- **DRU1** は 2 台のマシン、SYS1 と SYS3 で構成されます。HADB ノードのアクティブ 1 はマシン SYS1 上にあります。HADB ノードのスペア 3 はマシン SYS3 上にあります。

共存トポロジのバリエーション

スケーラビリティーとスループットを向上させるには、より多くのマシンを追加して、Application Server インスタンスと HADB ノードの数を増やします。たとえば、それぞれ 1 つの Application Server インスタンスと 1 つの HADB ノードを含む、2 台のマシンを追加できます。HADB ノードは必ずペアで追加し、各 DRU に対して 1 つのノードを割り当ててください。77 ページの「共存トポロジのバリエーション」は、この構成を示しています。

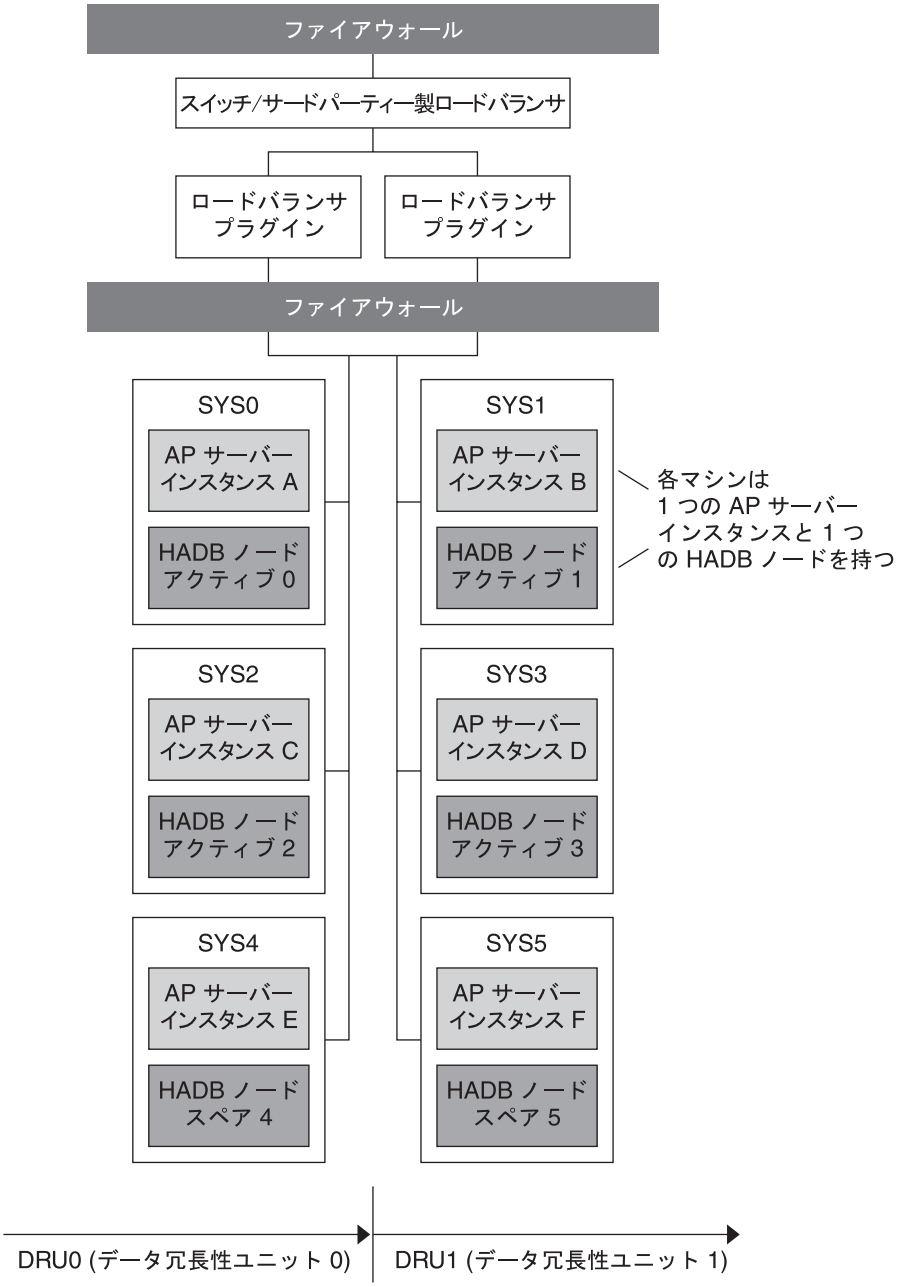


図 3–2 共存トポロジのバリエーション

このバリエーションでは、75 ページの「構成例」で説明された共存トポロジにマシン SYS4 と SYS5 が追加されています。

Application Server インスタンスは、次のようにホストされています。

- マシン SYS0 はインスタンス A をホストしている
- マシン SYS1 はインスタンス B をホストしている
- マシン SYS2 はインスタンス C をホストしている
- マシン SYS3 はインスタンス D をホストしている
- マシン SYS4 はインスタンス E をホストしている
- マシン SYS5 はインスタンス F をホストしている

これらのインスタンスは、情報を次の 2 つの DRU に持続させるクラスタを形成します。

- **DRU0** は、マシン SYS0、SYS2、および SYS4 で構成されます。HADB ノードのアクティブ 0 はマシン SYS0 上にあります。HADB ノードのアクティブ 2 はマシン SYS2 上にあります。HADB ノードのスペア 4 はマシン SYS4 上にあります。
- **DRU1** は、マシン SYS1、SYS3、および SYS5 で構成されます。HADB ノードのアクティブ 1 はマシン SYS1 上にあります。HADB ノードのアクティブ 3 はマシン SYS3 上にあります。HADB ノードのスペア 5 はマシン SYS5 上にあります。

分離層トポロジ

このトポロジでは、Application Server インスタンスと HADB ノードが別のマシン上に存在します。このため、 分離層 と呼ばれています。

このトポロジには、共存トポロジに比べて多くのハードウェアが必要です。このトポロジは、異なる種類のマシンがある場合に適している可能性があります。Application Server インスタンスをホストするためにあるマシンのセットを割り当て、HADB ノードをホストするために別のセットを割り当てることができます。たとえば、より性能の高いマシンを Application Server インスタンスに使用し、ほかのマシンを HADB に使用することもできます。

構成例

次の図は、分離層トポロジを示しています。

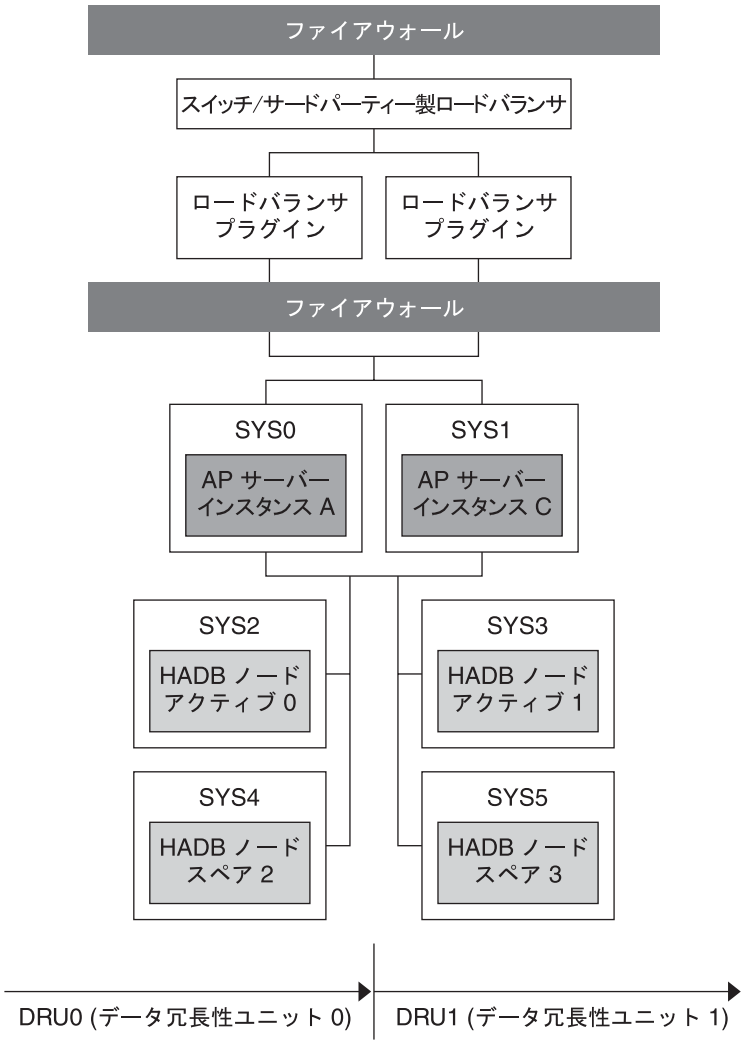


図3–3 分離層トポロジの例

このトポロジでは、マシン SYS0 は Application Server インスタンス A を、マシン SYS1 は Application Server インスタンス B をそれぞれホストしています。これらの 2 つのインスタンスは、セッション情報を次の 2 つの DRU に持続させるクラスタを形成します。

- **DRU0** は 2 台のマシン、SYS2 と SYS4 で構成されます。HADB ノードのアクティブ 0 はマシン SYS2 上にあり、HADB ノードのスペア 2 はマシン SYS4 上にあります。
- **DRU1** は 2 台のマシン、SYS3 と SYS5 で構成されます。HADB ノードのアクティブ 1 はマシン SYS3 上にあり、HADB ノードのスペア 3 はマシン SYS5 上にあります。

あるマシンに障害が発生した場合でも、任意の DRU の完全なデータが引き続きほかのマシンに使用可能になるように、DRU 上のすべてのノードが異なるマシン上にあります。

分離層トポロジのバリエーション

分離層トポロジのバリエーションとして、構成に対して水平方向により多くのマシンを追加することにより、Application Server インスタンスの数を増やします。たとえば、新しい Application Server インスタンスを作成することにより、構成例に別のマシンを追加します。同様に、HADB ノードをホストするためのマシンを追加することにより、HADB ノードの数を増やします。HADB ノードは、各 DRU に対して 1 つずつ、ペアで追加する必要があることに注意してください。

81 ページの「分離層トポロジのバリエーション」は、この構成を示しています。

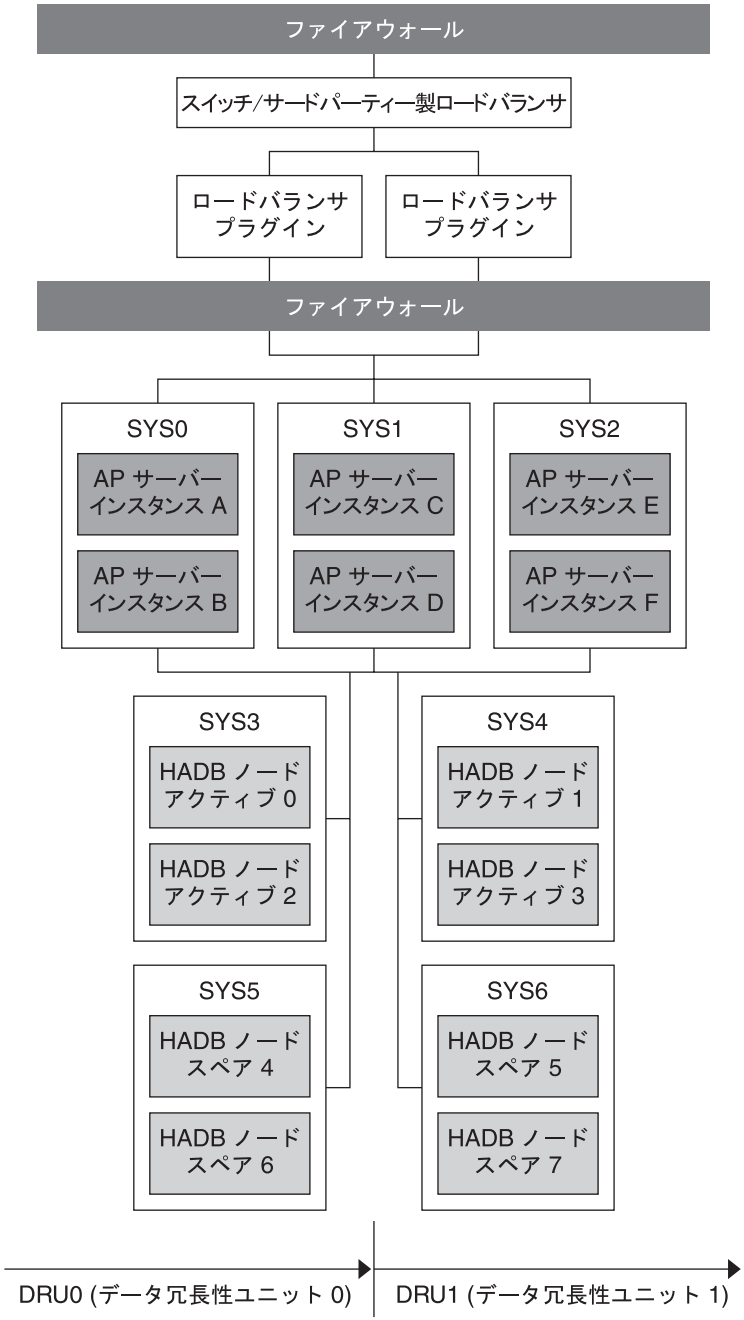


図3–4 分離層トポロジのバリエーション

この構成では、Application Server インスタンスをホストしている各マシンに 2 つのインスタンスがあります。したがって、クラスタ内には合計 6 つの Application Server インスタンスが存在します。

HADB ノードは、マシン SYS3、SYS4、SYS5、および SYS6 上にあります。

DRU0 は、次の 2 台のマシンで構成されます。

- HADB ノードのアクティブ 0 と HADB ノードのアクティブ 2 をホストしている SYS3。
- HADB ノードのスペア 4 と HADB ノードのスペア 6 を含む SYS5。

DRU1 は、次の 2 台のマシンで構成されます。

- HADB ノードのアクティブ 1 と HADB ノードのアクティブ 3 をホストしている SYS4。
- HADB ノードのスペア 5 と HADB ノードのスペア 7 をホストしている SYS6。

HADB ノードをホストしている各マシンが 2 つのノードをホストしています。したがって、4 つのアクティブノードと 4 つのスペアノードの、合計 8 つの HADB ノードが存在します。

使用するトポロジの決定

どのトポロジ (またはバリエーション) がパフォーマンスと可用性の要件をもっともよく満たすかを判断するには、各トポロジをテストし、マシンや CPU の異なる組み合わせで試してください。

目標を満たすには、どういうトレードオフが必要かを判断します。たとえば、保守の容易性が重要な場合は、分離層トポロジの方が適しています。この場合のトレードオフは、このトポロジには共存トポロジに比べて多くのマシンが必要なことです。

トポロジを選択する上で重要な要因は、使用可能なマシンの種類です。システムに大規模な対称多重処理 (SMP) マシンが含まれている場合は、これらのマシンの処理能力をフルに活用できるため、共存トポロジが有力です。システムにさまざまな種類のマシンが含まれている場合は、Application Server 層と HADB 層に異なるマシンのセットを割り当てることができるため、分離層トポロジの方が有効です。たとえば、もっとも性能の高いマシンを Application Server 層に使用し、その他のマシンを HADB 層に使用することもできます。

トポロジの比較

次の表は、共存トポロジと分離層トポロジを比較しています。左の列にはトポロジの名前が示され、中央の列は各トポロジの長所を、右の列は各トポロジの短所をそれぞれ示しています。

表 3–1 トポロジの比較

トポロジ	長所	短所
共存トポロジ	必要なマシンの数が少ない。HADB ノードと Application Server インスタンスが同じ層に存在するため、追加の負荷を処理する各スペアノード上に Application Server インスタンスを作成できます。 CPU 使用率の向上。1 台のマシンを共有する Application Server インスタンスと HADB ノードの間で処理が均等に分散されます。 マシンの処理能力をフルに活用できるため、大規模な対称多重処理 (SMP) マシンに有効。	保守がより複雑になる。たとえば、保守を実行するために HADB ノードをホストしているマシンをシャットダウンする必要がある場合は、そのマシン上のアプリケーションサーバーインスタンスも使用不可になります。
分離層トポロジ	保守が容易。たとえば、HADB ノードを停止することなく、Application Server インスタンスをホストしているマシン上で保守を実行できます。 異なる種類のマシンがある場合に有効。Application Server 層と HADB 層に異なるマシンのセットを割り当てることができます。たとえば、より性能の高いマシンを Application Server 層に使用し、その他のマシンを HADB 層に使用することができます。	共存トポロジに比べて多くのマシンが必要。アプリケーションサーバーインスタンスと HADB ノードが別の層に配置されるため、HADB スペアノードをホストしているマシンにはアプリケーションサーバーインスタンスを配置できません。 CPU 使用率の低下。アプリケーションサーバー層と HADB 層の負荷が不均一になる可能性があります。この問題は、マシンの台数が少ない (4 〜 6 台) 場合により大きくなります。

◆ ◆ ◆ 第 4 章

4 配備のためのチェックリスト

この付録では、Application Server を使用した評価および本番運用を始めるためのチェックリストを示します。

配備のためのチェックリスト

表 4–1 チェックリスト

コンポーネント/機能	説明
アプリケーション	配備するアプリケーションについて、次の要件を決定します。 ■ 必要または許容可能な応答時間。 ■ ピーク負荷特性。 ■ 必要な持続性の範囲と頻度。 ■ `web.xml` でのセッションタイムアウト設定。 ■ フェイルオーバーおよび可用性の要件。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 パフォーマンスチューニングガイド』を参照してください。
ハードウェア	■ HADB ノードのホストには、同じ種類のハードウェアを使用します。 ■ 必要なハードディスク領域およびメモリー容量が備わっています。 ■ サイズ決定の演習を使用して、配備の要件を識別します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 リリースノート (UNIX 版)』を参照してください。

表 4–1 チェックリスト *(続き)*

コンポーネント/機能	説明
オペレーティングシステム	■ 製品がサポート対象のプラットフォームにインストールされていることを確認します。 ■ パッチレベルが最新かつ適切であることを確認します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 リリースノート (UNIX 版)』を参照してください。
ネットワークインフラストラクチャー	■ シングルポイント障害を識別して対処します。 ■ NIC およびその他のネットワークコンポーネントが正しく設定されていることを確認します。 ■ `ttcp` ベンチマークテストを実行し、スループットが要件または期待値を満たしているかどうかを調べます。 ■ HADB ノードが正しくインストールされるように、必要に応じて `rsh/ssh` を設定します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 Installation Guide』を参照してください。
バックエンドおよびその他の外部データソース	ドメインの専門家またはベンダーの協力を得て、これらのデータソースが適切に設定されていることを確認します。
システムの変更/設定	■ パフォーマンステストまたはストレステストを実行する前に、`/etc/system` および Linux でこれに相当するファイルの変更が完了していることを確認します。 ■ TCP/IP 設定の変更が完了していることを確認します。 ■ システムのデフォルトでは、多くのサービスが事前に設定されています。これらすべてのサービスが実行のために必要とは限りません。システムリソースを節約するために、必要でないサービスは無効にします。 ■ Solaris では、`Setoolkit` を使用してシステムの動作を調べます。出現したフラグをすべて解決します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 パフォーマンスチューニングガイド』を参照してください。
Application Server および **HADB** のインストール	■ NFS マウントされたボリュームにこれらのサーバーをインストールしていないことを確認します。 ■ Application Server ノードと HADB ノードの両方を同じマシンにインストールするときは、十分なディスク領域および RAM があることを確認します。 ■ 複数の HADB ノードを同じシステムにインストールするときは、十分な独立ディスクがあることを確認します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 2 章「高可用性 (HA) データベースのインストールと設定」を参照してください。

表 4–1 チェックリスト (続き)

コンポーネント/機能	説明
HADB の設定	■ HADB データデバイスのサイズを設定します。 ■ DataBufferPoolSize を定義します。 ■ LogBufferSize を定義します。 ■ InternalBufferSize を定義します。 ■ NumberOfLocks を設定します。 ■ 各種の Application Server コンポーネントに対し、最適なタイムアウト値を設定します。 ■ ファイルシステム上に、HADB ノードの物理レイアウトを作成します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「HADB の設定」を参照してください。
Application Server の設定	■ ログ: アクセスログのローテーションを有効にします。 ■ 適切なログレベルを選択します。通常は WARNING が適切です。 ■ 管理コンソールを使用して J2EE コンテナを設定します。 ■ 管理コンソールを使用して HTTP リスナーを設定します。 ■ 管理コンソールを使用して ORB スレッドプールを設定します。 ■ ネイティブコードを必要とする Type2 ドライバまたは呼び出しを使用している場合、LD_LIBRARY_PATH に mtmalloc.so が指定されていることを確認します。 ■ 適切な持続性の範囲および頻度を使用しており、これらが個別の Web/EJB モジュールでオーバーライドされないことを確認します。 ■ SFSB の重要なメソッドのみがチェックポイント設定されることを確認します。 チューニングの詳細は、『Sun Java System Application Server Enterprise Edition 8.2 パフォーマンスチューニングガイド』を参照してください。 設定の詳細は、『Sun Java System Application Server Enterprise Edition 8.2 管理ガイド』を参照してください。
ロードバランサの設定	■ Web Server がインストールされていることを確認します。 ■ Web Server のロードバランサプラグインがインストールされていることを確認します。 ■ パッチチェックが無効なことを確認します。 ■ `KeepAliveQuery` パラメータの値を小さくします。負荷の低いシステムでは、この値が小さいほど待ち時間が短縮されます。負荷の高いシステムでは、この値が大きいほどスループットが高くなります。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 パフォーマンスチューニングガイド』の「キープアライブ」を参照してください。

表 4–1 チェックリスト (続き)

コンポーネント/機能	説明
Java 仮想マシンの設定	■ 最初は、最小および最大のヒープサイズを同じ値に設定し、各インスタンスには 1G バイト以上を設定します。 ■ 詳細は、 Java Hotspot VM Optionsを参照してください。 ■ Application Server の複数のインスタンスを実行するときは、プロセッサセットの作成を検討し、Application Server をそのセットにバインドします。これは、古い世代のスイープに CMS コレクタが使用される場合に効果的です。
ロードバランサのタイムアウト設定	■ Response-time-out-in-seconds - Application Server インスタンスが正常でないと宣言するまでにロードバランサが待機する時間。この値は、アプリケーションの応答時間に基づいて設定します。この値が大きすぎると、Web Server およびロードバランサプラグインは、Application Server インスタンスが正常でないと指定するまで長時間待機します。この値が小さすぎる場合に Application Server の応答時間がこのしきい値を超えると、インスタンスが正常でないという誤った指定がなされることになります。 ■ Interval-in-seconds – インスタンスが正常に戻ったかどうかを確認するために、正常でないインスタンスをチェックする間隔 (秒)。この値が小さすぎると、ロードバランサプラグインから Application Server インスタンスへの不必要なトラフィックが発生します。値が大きすぎると、正常に戻ったインスタンスへの要求のルーティングが遅れます。 ■ Timeout-in-seconds - 健全性検査要求に対して応答が取得される間隔。健全性検査が確実に成功するように、クラスタ内のシステム間のトラフィックに基づいてこの値を調整します。 詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の第 5 章「HTTP 負荷分散の設定」を参照してください。

表 4–1 チェックリスト (続き)

コンポーネント/機能	説明
HADB のタイムアウト設定	■ sql_client_timeout - アイドル状態のクライアントに対する SQLSUB の待ち時間。たとえば、ログオンしたクライアントは何らかの要求を送信したあと、ユーザー入力を待機します。アイドル状態が 30 分を超えたクライアントは死んでいるものとみなされ、セッションは終了されます。この値を小さく設定しすぎると、SQL セッションが完了する前に終了させてしまう可能性があります。この値を大きく設定しすぎると、アイドル状態ではないがすでに終了している SQL セッションがリソースを占有することがあります。その結果として、ほかの SQL クライアントのログオンが妨げられる可能性があります。この変数を調整するときは、`nsessions` の設定も考慮します。HADB の JDBC 接続プールの steady-pool-size が max-pool-size よりも大きい場合、idle-timeout-in-seconds を sql_client_timeout よりも小さく設定して、HADB が接続を閉じる前に Application Server 自体が接続を閉じるように設定できます。デフォルト値は 1800 秒です。 ■ lock_timeout - トランザクションがデータへのアクセスを待機する最大時間 (ミリ秒)。この時間を超過すると、トランザクションは「The transaction timed out.」というエラーメッセージを生成します。このようなタイムアウトは、ほかのトランザクションが保持しているロック (デッドロック) を待機するトランザクションが原因で発生し、サーバー負荷を上昇させます。この値を 500 ミリ秒未満に設定しないでください。サーバーログに「transaction timed out」というメッセージが見つかった場合は、この値を大きくします。ロックのタイムアウト値を設定するには、HADB の JDBC 接続プールに `<property name=lockTimeout value="x"\>` というプロパティーを追加します。デフォルト値は 500 ミリ秒です。 ■ Querytimeout - クエリーの実行を HADB が待機する最大時間 (ミリ秒)。クエリーのタイムアウトを示す例外がサーバーログに頻繁に記録されている場合は、この値を大きくすることを検討します。この値を設定するには、HADB の JDBC 接続プールに `<property name=QueryTimeout value="x"\>` というプロパティーを追加します。デフォルト値は 30 秒です。 ■ loginTimeout - クライアントが HADB へのログインを待機する最大時間 (秒)。この値を設定するには、HADB の JDBC 接続プールに `<property name=loginTimeout value="x"\>` というプロパティーを追加します。デフォルト値は 10 秒です。 ■ MaxTransIdle - クライアントに応答を送信してから次の要求を受信する間に、トランザクションがアイドル状態でいられる最大時間 (ミリ秒)。この値を変更するには、HADB の JDBC 接続プールに `<property name=maxtransIdle value="x"\>` というプロパティーを追加します。デフォルト値は 40 秒です。 詳細は、『Sun Java System Application Server パフォーマンスチューニングガイド』を参照してください。

表 4–1 チェックリスト *(続き)*

コンポーネント/機能	説明
Application Server のタイムアウト設定	■ Max-wait-time-millis - プールからの接続取得を待機し、超過した場合に例外をスローする時間。デフォルトは 6 秒です。持続されるデータのサイズが 50K バイトを超える高負荷システムでは、この値の変更を検討します。 ■ Cache-idle-timeout-in-seconds - 非活性化されるまでに EJB がキャッシュ内でアイドル状態でいられる時間。エンティティー Bean およびステートフルセッション Bean のみに適用されます。 ■ Removal-timeout-in-seconds - EJB が非活性化状態 (バックアップストア内でアイドル状態) にとどまる時間。デフォルト値は 60 分です。この値は、SFSB フェイルオーバーの必要性に基づいて調整します。 これらすべての値を調整するときは、HADB の JDBC 接続プールの max-wait-time-in-millis の設定に注意を払います。詳細は、『Sun Java System Application Server Enterprise Edition 8.2 高可用性 (HA) 管理ガイド』の「JDBC 接続プールの設定」を参照してください。
VM ガベージコレクション **(GC)** のチューニング	ガベージコレクションが 4 秒以上一時停止すると、セッション状態を HADB に持続する際に一時的な問題の原因となる可能性があります。この問題を避けるには、VM ヒープを調整します。データ持続の失敗を 1 回も許容できない場合や、システムが高負荷状態でないときは、CMS コレクタまたはスループットコレクタを使用します。 これらのコレクタを有効にするには、次のオプションを追加します。 `<jvm-options>-XX:+UseConcMarkSweepGC</jvm-options>` このオプションはスループットを低下させる場合があります。

索引

る

ろ