Sun Java System Application Server 9.1 배포 계획 설명서

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

부품 번호: 820–4902

080429@19860

목차

그림

표

코드 예

머리말

배포 계획 설명서에서는 프로덕션 배포를 구축하는 방법에 대해 설명합니다.

이 장에서는 전체 Sun Java™ System Application Server 설명서 모음에 대한 정보와 규칙이 제공됩니다.

Application Server 설명서 모음

Application Server 설명서 모음에서는 배포 계획 및 시스템 설치에 대해 설명합니다. Application Server 설명서에 대한 URL(Uniform Resource Locator)은 `http://docs.sun.com/coll/1343.4` 및 `http://docs.sun.com/coll/1776.1`입니다. Application Server 소개 내용을 보려면 다음 표에 나열된 설명서를 순서대로 참조하십시오.

표 P–1 Application Server 설명서 모음에 포함된 설명서

설명서 제목	설명
Documentation Center(설명서 센터)	작업과 주제로 구성된 Application Server 설명서 항목입니다.
릴리스 노트	소프트웨어 및 설명서 관련 최신 정보로 지원되는 하드웨어, 운영 체제, JDK™(Java Development Kit) 및 데이터베이스 드라이버를 표를 기반으로 종합적으로 요약하였습니다.
빠른 시작 설명서	Application Server 제품을 시작하는 방법에 대해 설명합니다.
설치 설명서	소프트웨어와 해당 구성 요소 설치에 대해 설명합니다.
배포 계획 설명서	사용자 시스템 요구 사항과 기업 평가를 통해 Application Server를 사용자 사이트에 가장 적합한 방식으로 배포하는 방법에 대해 설명합니다. 서버 배포 시 알아야 할 일반적인 문제와 관심을 기울여야 할 사항에 대해서도 설명합니다.
Application Deployment Guide	응용 프로그램과 응용 프로그램 구성 요소를 Application Server에 배포하는 방법에 대해 설명합니다. 배포 설명자에 대한 정보를 제공합니다.

표 P–1 Application Server 설명서 모음에 포함된 설명서 *(계속)*

설명서 제목	설명
Developer's Guide	Java EE 구성 요소 및 API용 개방형 Java 표준 모델을 따르는 Application Server에서 실행할 Java Platform, Enterprise Edition(Java EE 플랫폼) 응용 프로그램을 만들고 구현하는 방법에 대해 설명합니다. 개발자 도구, 보안, 디버깅 및 라이프사이클 모듈 생성에 대한 정보를 제공합니다.
Java EE 5 Tutorial	Java EE 5 플랫폼 기술과 API를 사용하여 Java EE 응용 프로그램을 개발하는 방법에 대해 설명합니다.
Java WSIT Tutorial	WSIT(Web Service Interoperability Technologies)를 사용하여 웹 응용 프로그램을 개발하는 방법에 대해 설명합니다. WSIT 기술과 각 기술이 지원하는 기능 및 옵션을 사용하는 방법, 시기 및 이유에 대해 설명합니다.
관리 설명서	구성, 모니터링, 보안, 자원 관리 및 웹 서비스 관리를 비롯한 Application Server 시스템 관리에 대해 설명합니다.
고가용성 관리 설명서	고가용성 데이터베이스를 위한 설치 후 구성 및 관리 방법에 대해 설명합니다.
Administration Reference	Application Server 구성 파일인 `domain.xml`을 편집하는 방법에 대해 설명합니다.
Upgrade and Migration Guide	Application Server의 이전 버전에서 업그레이드하거나 경쟁 응용 프로그램 서버에서 Java EE 응용 프로그램을 마이그레이션하는 방법에 대해 설명합니다. 제품 사양과 호환되지 않는 결과를 가져올 수 있는 제품 릴리스 및 구성 옵션의 차이점에 대한 설명도 포함되어 있습니다.
Performance Tuning Guide	Application Server를 조정하여 성능을 향상시키는 방법에 대해 설명합니다.
Troubleshooting Guide	Application Server 문제를 해결하는 방법에 대해 설명합니다.
Error Message Reference	Application Server 오류 메시지를 해결하는 방법에 대해 설명합니다.
Reference Manual	설명서 페이지 스타일로 작성되었으며, Application Server와 같이 사용할 수 있는 유틸리티 명령에 대해 설명합니다. `asadmin` 명령줄 인터페이스를 포함합니다.

관련 설명서

Application Server는 단독으로 구입하거나 네트워크 또는 인터넷 환경에서 배포된 엔터프라이즈 응용 프로그램을 지원하는 소프트웨어 인프라인 Sun Java Enterprise System(Java ES)의 구성 요소로 구입할 수 있습니다. Application Server를 Java ES의 구성 요소로 구입한 경우에는 `http://docs.sun.com/coll/1286.2` 및 `http://docs.sun.com/coll/1397.2`의 시스템 설명서를 잘 이해해야 합니다. Java ES 및 해당 구성 요소에 대한 모든 설명서의 URL은 `http://docs.sun.com/prod/entsys.5`입니다.

다른 독립 실행형 Sun Java System 서버 제품에 대한 설명서는 다음 위치에서 참조하십시오.

- Message Queue 설명서(`http://docs.sun.com/coll/1343.4` 및 `http://docs.sun.com/coll/1776.1`)

- Directory Server 설명서(http://docs.sun.com/coll/1224.1 및 http://docs.sun.com/coll/1586.1)
- Web Server 설명서(http://docs.sun.com/coll/1308.3 및 http://docs.sun.com/coll/1410.2)

Application Server와 함께 제공되는 패키지에 대한 Javadoc 도구 참조는 http://glassfish.dev.java.net/nonav/javaee5/api/index.html에 있습니다. 또한 다음과 같은 자원을 사용할 수 있습니다.

- Java EE 5 사양 (http://java.sun.com/javaee/5/javatech.html)
- Java EE 5 자습서 (http://java.sun.com/javaee/5/docs/tutorial/doc/index.html)
- Java EE Blueprints (http://java.sun.com/reference/blueprints/index.html)

기본 경로 및 파일 이름

다음 표에서는 본 설명서에서 사용한 기본 경로와 파일 이름에 대해 설명합니다.

표 P–2 기본 경로 및 파일 이름

자리 표시자	설명	기본값

install-dir	Application Server의 기본 설치 디렉토리를 나타냅니다.	Solaris™ 운영 체제에 Java ES를 설치한 경우:
		`/opt/SUNWappserver/appserver`
		Linux 운영 체제에 Java ES를 설치한 경우:
		`/opt/sun/appserver/`
		기타 Solaris 및 Linux 설치, 루트가 아닌 사용자의 경우:
		user's-home-directory`/SUNWappserver`
		기타 Solaris 및 Linux 설치, 루트 사용자의 경우:
		`/opt/SUNWappserver`
		Windows, 모든 설치:
		SystemDrive`:\Sun\AppServer`

domain-root-dir	모든 도메인을 포함한 디렉토리를 나타냅니다.	Java ES Solaris 설치:
		`/var/opt/SUNWappserver/domains/`
		Java ES Linux 설치:
		`/var/opt/sun/appserver/domains/`
		모든 기타 설치:
		install-dir`/domains/`

표 P–2 기본 경로 및 파일 이름 *(계속)*

자리 표시자	설명	기본값
domain-dir	도메인용 디렉토리를 나타냅니다. 구성 파일에서 *domain-dir*이 다음과 같이 표시되어 있을 수 있습니다. `${com.sun.aas.instanceRoot}`	*domain-root-dir*/*domain-dir*
instance-dir	서버 인스턴스용 디렉토리를 나타냅니다.	*domain-dir*/*instance-dir*

활자체 규약

다음 표에서는 본 설명서에 사용된 활자체 규약에 대해 설명합니다.

표 P–3 활자체 규약

서체	의미	예
`AaBbCc123`	명령, 파일 및 디렉토리의 이름과 컴퓨터 화면에 출력되는 내용입니다.	`.login` 파일을 편집합니다. `ls -a`를 사용하여 모든 파일을 나열합니다. `machine_name% you have mail.`
`AaBbCc123`	화면 상의 컴퓨터 출력과는 반대로 사용자가 직접 입력하는 사항입니다.	`machine_name%` **`su`** `Password:`
AaBbCc123	명령줄 자리 표시자: 실제 이름이나 값으로 대체됩니다.	파일을 삭제하려면 `rm` *filename*을 입력하십시오.
AaBbCc123	책 제목, 새로 나오는 용어, 강조 표시할 단어입니다. (강조 표시된 일부 항목은 온라인에서 볼드로 표시됩니다.)	**사용자 설명서**의 6장을 참조하십시오. *cache*는 로컬로 저장된 복사본입니다. 파일을 저장하지 **마십시오**.

기호 규칙

다음 표에서는 본 설명서에 사용된 기호에 대해 설명합니다.

표 P–4 기호 규칙

기호	설명	예	의미
`[ ]`	선택 인수 및 명령 옵션을 포함합니다.	`ls [-l]`	`-l` 옵션은 사용하지 않아도 됩니다.

표 P–4 기호 규칙 (계속)

기호	설명	예	의미
{ | }	필수 명령 옵션에 대한 일련의 선택 항목을 포함합니다.	`-d {y|n}`	`-d` 옵션에서는 `y` 인수나 `n` 인수를 사용해야 합니다.
${ }	변수 참조를 나타냅니다.	`${com.sun.javaRoot}`	`com.sun.javaRoot` 변수 값을 참조합니다.
-	동시에 입력하는 여러 키를 결합합니다.	Control-A	Ctrl 키를 누른 채로 A 키를 누릅니다.
+	연속해서 입력하는 여러 키를 결합합니다.	Ctrl+A+N	Ctrl 키를 눌렀다가 놓은 다음 후속 키를 누릅니다.
→	그래픽 사용자 인터페이스의 메뉴 항목 선택을 나타냅니다.	파일 → 새로 만들기 → 템플리트	파일 메뉴에서 새로 만들기를 선택합니다. 새로 만들기 하위 메뉴에서 템플리트를 선택합니다.

설명서, 지원 및 교육

Sun 웹 사이트에서는 다음 추가 자원에 대한 정보가 제공됩니다.

- 설명서(`http://www.sun.com/documentation/`)
- 지원(`http://www.sun.com/support/`)
- 교육(`http://www.sun.com/training/`)

Sun 제품 설명서 검색

docs.sun.com℠ 웹 사이트에서 Sun 제품 설명서를 검색하는 방법 이외에, 검색 필드에 다음 구문을 입력하여 검색 엔진을 사용하는 방법도 있습니다.

search-term `site:docs.sun.com`

예를 들어 "broker"를 검색하려면 다음을 입력합니다.

```
broker site:docs.sun.com
```

다른 Sun 웹 사이트를 검색에 포함하려면(예: java.sun.com, www.sun.com 및 developers.sun.com) 검색 필드에서 `docs.sun.com` 대신 `sun.com`을 사용합니다.

타사 웹 사이트 참조 사항

이 설명서에서는 추가 관련 정보를 제공하기 위해 타사 URL을 참조하기도 합니다.

주 – Sun은 이 설명서에 언급된 타사 웹 사이트의 가용성에 대해 책임지지 않습니다. Sun은 이러한 사이트나 자원을 통해 사용할 수 있는 내용, 광고, 제품 또는 기타 자료에 대해서는 보증하지 않으며 책임지지 않습니다. Sun은 해당 사이트 또는 자원을 통해 사용 가능한 내용, 제품 또는 서비스의 사용과 관련해 발생하거나 발생했다고 간주되는 손해나 손실에 대해 책임이나 의무를 지지 않습니다.

사용자 의견 환영

Sun은 설명서의 내용을 지속적으로 개선하고자 하며 사용자 여러분의 의견과 제안을 환영합니다. 사용자 의견을 보내시려면 `http://docs.sun.com`에서 의견 보내기를 누르십시오. 온라인 양식에서 전체 문서 제목과 부품 번호를 기입해 주십시오. 부품 번호는 해당 설명서의 제목 페이지나 문서의 URL에 있으며 7자리 또는 9자리 숫자로 되어 있습니다. 예를 들어, 이 설명서의 부품 번호는 820-4902입니다.

◆ ◆ ◆ 1 장

1 제품 개념

Sun Java System Application Server는 J2EE 응용 프로그램의 개발, 배포 및 관리를 위한 견고한 플랫폼을 제공합니다. 주요 기능으로는 확장 가능한 트랜잭션 관리, 컨테이너 관리 지속성 런타임, 웹 서비스 성능, 클러스터링, 고가용성 세션 상태, 보안 및 통합 기능이 있습니다.

이 장은 다음 내용으로 구성되어 있습니다.

J2EE 플랫폼 개요

Application Server는 J2EE(Java 2 Enterprise Edition) 1.4 기술을 구현합니다. J2EE 플랫폼은 Application Server의 응용 프로그램 구성 요소, API 및 런타임 컨테이너와 서비스를 설명하는 표준 사양 집합입니다.

J2EE 응용 프로그램

J2EE 응용 프로그램은 JSP(JavaServer Pages), Java Servlet 및 EJB(Enterprise JavaBeans)와 같은 구성 요소로 구성됩니다. 소프트웨어 개발자는 이러한 구성 요소를 사용하여 대규모 분산 응용 프로그램을 구축할 수 있습니다. 개발자는 J2EE 응용 프로그램을 작업 사이트에 배포할 수 있는 JAR(Java ARchive) 파일(zip 파일과 유사)로 패키지화합니다. 관리자는 J2EE JAR 파일을 하나 이상의 서버 인스턴스(또는 인스턴스 클러스터)에 배포하여 J2EE 응용 프로그램을 Application Server에 설치합니다.

다음 그림은 다음 절에 설명된 J2EE 플랫폼 구성 요소를 보여줍니다.

죄송합니다. 현재 그래픽을 사용할 수 없습니다.

컨테이너

각 서버 인스턴스에는 웹 및 EJB의 두 가지 컨테이너가 있습니다. 컨테이너는 J2EE 구성 요소에 보안 및 트랜잭션 관리 등의 서비스를 제공하는 런타임 환경입니다. Java Server Pages 및 서블릿 같은 웹 구성 요소는 웹 컨테이너에서 실행됩니다. Enterprise JavaBeans는 EJB 컨테이너에서 실행됩니다.

J2EE 서비스

J2EE 플랫폼은 다음과 같은 응용 프로그램 서비스를 제공합니다.

- **이름 지정** - 이름 지정 및 디렉토리 서비스는 객체를 이름에 바인드합니다. J2EE 응용 프로그램은 해당 JNDI(Java Naming and Directory Interface) 이름을 조회하여 객체를 찾을 수 있습니다.
- **보안** - JACC(Java Authorization Contract for Containers)는 J2EE 컨테이너에 대해 정의된 보안 계약 집합입니다. 클라이언트의 아이디에 따라 컨테이너에서 컨테이너의 자원 및 서비스에 대한 액세스를 제한할 수 있습니다.
- **트랜잭션 관리** - 트랜잭션은 가장 작은 작업 단위입니다. 예를 들어, 은행 계좌 간에 자금을 이체하는 것이 트랜잭션입니다. 트랜잭션 관리 서비스는 트랜잭션이 완료되거나 롤백되는 것을 보장합니다.
- 메시지 서비스 - 각각 별개의 시스템에서 호스팅되는 응용 프로그램은 JMS(Java™ Message Service)를 통해 메시지를 교환하는 방식으로 서로 통신할 수 있습니다. JMS는 J2EE 플랫폼의 필수 요소로서, 이종 엔터프라이즈 응용 프로그램의 통합 작업을 간소화합니다.

웹 서비스

클라이언트는 J2EE 1.4 응용 프로그램에 HTTP, RMI/IIOP 및 JMS를 통해 액세스할 뿐 아니라 원격 웹 서비스로 액세스할 수 있습니다. 웹 서비스는 JAX-RPC(Java API for XML-based RPC)를 사용하여 구현됩니다. J2EE 응용 프로그램은 네트워크 응용 프로그램에서 일반적인, 웹 서비스에 대한 클라이언트 역할을 담당할 수도 있습니다.

WSDL(Web Services Description Language)은 웹 서비스 인터페이스를 설명하는 XML 형식입니다. 웹 서비스 사용자는 WSDL 문서를 동적으로 구문 분석하여 웹 서비스에서 제공하는 작업과 작업 수행 방법을 확인합니다. Application Server는 다른 응용 프로그램에서 JAXR(Java API for XML Registries)을 통해 액세스할 수 있는 레지스트리를 사용하여 웹 서비스 인터페이스 설명을 배포합니다.

클라이언트 액세스

클라이언트는 다양한 방법으로 J2EE 응용 프로그램에 액세스할 수 있습니다. 브라우저 클라이언트는 HTTP(HyperText Transfer Protocol)를 사용하여 웹 응용 프로그램에 액세스합니다. 브라우저에는 보안 통신을 위해 SSL(Secure Sockets Layer)을 사용하는 HTTPS(HTTP Secure) 프로토콜이 사용됩니다.

Application Client Container에서 실행되는 다양한 클라이언트 응용 프로그램은 ORB(Object Request Broker), RMI(Remote Method Invocation), 그리고 IIOP(internet inter-ORB Protocol) 또는 IIOP/SSL(보안 IIOP)을 사용하여 Enterprise JavaBeans를 직접 조회 및 액세스할 수 있습니다. 또한, HTTP/HTTPS, JMS 및 JAX-RPC를 사용하여 응용 프로그램과 웹 서비스에 액세스할 수 있으며 JMS를 사용하여 응용 프로그램 및 Message-Driven Bean과 메시지를 주고 받을 수 있습니다.

WS-I(Web Services-Interoperability) 기본 프로필을 준수하는 클라이언트는 J2EE 웹 서비스에 액세스할 수 있습니다. WS-I는 J2EE 표준을 구성하는 필수 요소로서 상호 운용 가능한 웹 서비스를 정의하며 지원되는 언어로 작성된 클라이언트에서 Application Server에 배포된 웹 서비스에 액세스할 수 있도록 해줍니다.

최상의 액세스 메커니즘은 특정 응용 프로그램과 예상 트래픽 양에 따라 다릅니다. Application Server에서는 HTTP, HTTPS, JMS, IIOP 및 IIOP/SSL에 대해 개별적으로 구성할 수 있는 수신기를 지원합니다. 확장성과 안정성 향상을 위해 프로토콜별로 여러 수신기를 설정할 수 있습니다.

J2EE 응용 프로그램은 다른 서버에 배포된 Enterprise JavaBeans 모듈과 같은 J2EE 구성 요소의 클라이언트 역할을 할 수도 있고 이러한 모든 액세스 메커니즘을 사용할 수도 있습니다.

외부 시스템 및 자원

J2EE 플랫폼의 경우 외부 시스템을 **자원**이라고 합니다. 예를 들어 데이터베이스 관리 시스템은 JDBC 자원입니다. 각 자원은 JNDI(Java Naming and Directory Interface) 이름으로 고유하게 식별됩니다. 응용 프로그램은 다음 API와 구성 요소를 통해 외부 시스템에 액세스합니다.

- *Java Database Connectivity(JDBC)* - 데이터베이스 관리 시스템(Database Management System, DBMS)은 데이터를 저장, 구성 및 검색하는 기능을 제공합니다. 대부분의 비즈니스 응용 프로그램은 관계형 데이터베이스에 데이터를 저장합니다. 응용 프로그램은 JDBC를 통해 관계형 데이터베이스에 액세스합니다. Application Server에는 배포용으로는 적합하지 않지만 샘플 응용 프로그램과 응용 프로그램 개발 및 프로토타입 생성에 사용할 수 있는 PointBase DBMS가 포함되어 있습니다. Application Server는 주요 관계형 데이터베이스 연결에 사용할 수 있는 인증된 JDBC 드라이버를 제공합니다. 이러한 드라이버는 배포에 적합합니다.

- *Java Message* Service - 메시징은 소프트웨어 구성 요소 또는 응용 프로그램 간 통신 방법입니다. 메시징 클라이언트는 JMS(Java Messaging Service) API를 구현하는 메시징 공급자를 통해 다른 클라이언트와 메시지를 주고 받습니다. Application Server에는 고성능 JMS 브로커인 Sun Java System Message Queue가 포함되어 있습니다. Application Server의 Platform Edition에는 Message Queue의 무료 Platform Edition이 포함되어 있습니다. Application Server Enterprise Edition에는 클러스터링과 페일오버를 지원하는 Message Queue Enterprise Edition이 포함되어 있습니다.
- **J2EE 커넥터** - J2EE 커넥터 구조를 사용하여 J2EE 응용 프로그램과 기존 EIS(Enterprise Information Systems)를 통합할 수 있습니다. 응용 프로그램은 JDBC 드라이버를 사용하여 RDBMS에 액세스하는 것과 같이 **커넥터** 또는 **자원 어댑터**라는 이동 가능한 J2EE 구성 요소를 통해 EIS에 액세스합니다. 자원 어댑터는 독립 실행형 RAR(Resource Adapter Archive) 모듈로 배포되거나 J2EE 응용 프로그램 아카이브에 포함되어 있습니다. RAR로서 자원 어댑터는 다른 J2EE 구성 요소처럼 배포됩니다. Application Server에는 일반적인 EIS와 통합되는 평가 자원 어댑터가 있습니다.
- **JavaMail** - 응용 프로그램에서 JavaMail API를 통해 단순 메일 전송 프로토콜(Simple Mail Transport Protocol, SMTP) 서버에 연결하여 전자 메일을 주고 받을 수 있습니다.

Application Server 구성 요소

이 절에서는 다음과 같은 Sun Java System Application Server 구성 요소에 대해 설명합니다.

- 23 페이지 “서버 인스턴스”
- 23 페이지 “관리 도메인”
- 24 페이지 “클러스터”
- 24 페이지 “노드 에이전트”
- 25 페이지 “명명된 구성”
- 26 페이지 “HTTP 로드 밸런서 플러그인”
- 27 페이지 “클러스터의 IIOP 로드 균형 조정”
- 28 페이지 “메시지 대기열 및 JMS 자원”

다음 그림은 고가용성을 제공하는 단순 토폴로지 예를 통해 이러한 Application Server 구성 요소가 상호 작용하는 방법을 보여줍니다. 이 토폴로지 예에서는 한 관리자가 클러스터로 구성된 두 개의 시스템을 관리합니다. HADB 및 Application Server 프로세스는 동일한 시스템에 있습니다. Domain Administration Server는 단독으로 별도 시스템에서 호스팅되거나 Application Server 인스턴스를 호스팅하는 시스템 중 한 시스템에서 호스팅될 수 있습니다. 다이어그램의 선은 통신 또는 제어를 나타냅니다.

브라우저 기반 관리 콘솔과 같은 관리 도구는 DAS(Domain Administration Server)와 통신하고 DAS는 다시 노드 에이전트 및 서버 인스턴스와 통신합니다.

서버 인스턴스

서버 인스턴스는 단일 Java 가상 머신(Java Virtual Machine, JVM) 프로세스에서 실행되는 Application Server입니다. Application Server는 J2SE(Java 2 Standard Edition) 5.0 및 1.4를 사용하여 인증됩니다. 권장 J2SE 배포가 Application Server 설치에 포함되어 있습니다.

Application Server 및 함께 제공되는 JVM은 여러 프로세서로 확장 가능하도록 개발되었으므로 일반적으로 한 시스템에서 한 개의 서버 인스턴스를 만드는 것으로 충분합니다. 그러나 응용 프로그램 격리 및 롤링 업그레이드를 고려하면 한 시스템에서 여러 인스턴스를 만드는 것이 유리할 수 있습니다. 또한, 경우에 따라 여러 인스턴스가 포함된 대규모 서버는 관리 도메인 이상으로 사용될 수 있습니다. 관리 도구를 사용하면 여러 시스템에 걸쳐 서버 인스턴스를 간편하게 생성, 삭제 및 관리할 수 있습니다.

관리 도메인

관리 도메인(또는 **도메인**)은 함께 관리되는 서버 인스턴스의 그룹입니다. 서버 인스턴스는 단일 관리 도메인에 속합니다. 도메인의 인스턴스는 서로 다른 물리적 호스트에서 실행될 수 있습니다.

하나의 Application Server 설치에서 여러 도메인을 만들 수 있습니다. 서버 인스턴스를 도메인으로 그룹화하면 서로 다른 조직이나 관리자가 단일 Application Server 설치를 공유할 수 있습니다. 도메인마다 다른 도메인과 독립된 고유한 구성 로그 파일 및 응용 프로그램 배포 영역이 있습니다. 도메인 구성을 변경해도 다른 도메인의 구성에 영향을 미치지 않습니다. 마찬가지로, 응용 프로그램을 한 도메인에 배포해도 다른 도메인에 배포되거나 표시되지 않습니다. 관리자는 항상 하나의 도메인에만 인증되므로 해당 도메인에 대해서만 관리를 수행할 수 있습니다.

DAS(Domain Administration Server)

도메인에는 특수 설계된 Application Server 인스턴스로서, 관리 응용 프로그램을 호스팅하는 Domain Administration Server(DAS)가 있습니다. DAS는 관리자를 인증하고 관리 도구의 요청을 승인하며 도메인의 서버 인스턴스와 통신하여 요청을 수행합니다.

관리 도구에는 `asadmin` 명령줄 도구, 브라우저 기반 관리 콘솔이 있습니다. Application Server에서는 서버 관리용 JMX 기반 API도 제공합니다. 관리자가 한 번에 하나의 도메인을 보고 관리할 수 있으므로 안전하게 구분되어 관리됩니다.

DAS는 *admin* **서버** 또는 **기본 서버**라고도 합니다. DAS가 일부 관리 작업의 기본 대상이 되므로 기본 서버라고 합니다.

DAS는 Application Server 인스턴스이므로 테스트를 위해 J2EE 응용 프로그램을 호스팅할 수 있지만 DAS를 사용하여 프로덕션 응용 프로그램을 호스팅하지 마십시오. 예를 들어, 프로덕션 응용 프로그램을 호스팅할 클러스터와 인스턴스가 아직 생성되지 않은 경우 응용 프로그램을 DAS에 배포하려고 할 수 있습니다.

DAS는 도메인과 모든 배포된 응용 프로그램의 구성이 포함된 저장소를 유지합니다. DAS가 비활성화되거나 다운될 경우 활성 서버 인스턴스의 성능이나 가용성에는 영향이 없지만 관리 항목을 변경할 수 없습니다. 경우에 따라 보안을 위해 의도적으로 DAS 프로세스를 중지하는 것이 유용할 수 있습니다(예: 프로덕션 구성 중단).

도메인 구성과 응용 프로그램을 백업 및 복원하기 위한 관리 명령이 제공됩니다. 표준 백업 및 복원 절차를 사용하여 작업 구성을 신속하게 복원할 수 있습니다. DAS 호스트에 오류가 발생한 경우 이전 도메인 구성을 복원하려면 새 DAS 설치를 만들어야 합니다. 자세한 내용은 **Sun Java System Application Server 9.1 관리 설명서**의 "Domain Administration Server 다시 만들기"를 참조하십시오.

Sun Cluster 데이터 서비스는 DAS 호스트 IP 주소의 페일오버 및 전역 파일 시스템 사용을 통해 DAS 고가용성을 제공합니다. 이 솔루션은 거의 지속적인 DAS 가용성과 다양한 실패 유형에 대한 저장소를 제공합니다. Sun Cluster 데이터 서비스는 Sun Java Enterprise System과 함께 제공되거나 Sun Cluster와 함께 별도로 구입할 수 있습니다. 자세한 내용은 Sun Cluster 데이터 서비스에 대한 설명서를 참조하십시오.

클러스터

클러스터는 동일한 응용 프로그램, 자원 및 구성 정보를 공유하는 명명된 서버 인스턴스 모음입니다. 여러 시스템의 서버 인스턴스를 하나의 논리적 클러스터에 그룹화한 후 하나의 단위로 관리할 수 있습니다. DAS를 사용하여 다중 시스템 클러스터의 수명 주기를 쉽게 제어할 수 있습니다.

클러스터를 구축하면 수평적인 확장, 로드 균형 조정 및 페일오버 보호가 구현됩니다. 정의에 따르면 한 클러스터의 모든 인스턴스에 동일한 자원 및 응용 프로그램 구성이 포함됩니다. 클러스터의 서버 인스턴스나 시스템에 장애가 발생하면 로드 밸런서는 장애를 감지하고, 실패한 인스턴스에서 클러스터의 다른 인스턴스로 트래픽을 리디렉션하고, 사용자 세션 상태를 복구합니다. 동일한 응용 프로그램 및 자원가 클러스터의 모든 인스턴스에 있으므로 한 인스턴스에서 클러스터의 다른 인스턴스로 페일오버가 수행될 수 있습니다.

클러스터, 도메인 및 인스턴스는 다음과 같이 관련되어 있습니다.

- 관리 도메인은 0개 이상의 클러스터를 포함할 수 있습니다.
- 클러스터는 하나 이상의 서버 인스턴스를 포함할 수 있습니다.
- 클러스터는 단일 도메인에 속합니다.

노드 에이전트

노드 에이전트는 DAS를 호스팅하는 시스템을 비롯하여 서버 인스턴스를 호스팅하는 모든 시스템에서 실행되는 경량 프로세스입니다. 노드 에이전트의 기능은 다음과 같습니다.

- DAS의 명령에 따라 서버 인스턴스를 시작 및 중지합니다.

- 실패한 서버 인스턴스를 다시 시작합니다.
- 실패한 서버의 로그 파일 뷰를 제공하여 원격 진단을 지원합니다.
- 각 서버 인스턴스의 로컬 구성 저장소와 DAS의 중앙 저장소를 동기화하므로 노드 에이전트 관찰 하에 서버 인스턴스를 시작합니다.
- 인스턴스가 처음 생성될 때 인스턴스에 필요한 디렉토리를 만들고 인스턴스 구성과 중앙 저장소를 동기화합니다.
- 서버 인스턴스가 삭제될 때 적절한 정리를 수행합니다.

각 물리적 호스트에는 호스트가 속한 도메인별로 하나 이상의 노드 에이전트가 있어야 합니다. 물리적 호스트의 인스턴스가 둘 이상의 도메인에 포함된 경우 도메인별로 노드 에이전트가 필요합니다. 하나의 호스트에서 도메인별로 둘 이상의 노드 에이전트를 갖는 것은 가능하지만 아무런 이점이 없습니다.

노드 에이전트는 서버 인스턴스를 시작 및 중지하기 때문에 항상 실행 중이어야 합니다. 따라서 운영 체제가 부트될 때 시작됩니다. Solaris 및 기타 Unix 플랫폼의 경우 `inetd` 프로세스로 노드 에이전트를 시작할 수 있습니다. Windows의 경우 노드 에이전트를 Windows 서비스로 만들 수 있습니다.

노드 에이전트에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 8 장, “노드 에이전트 구성”을 참조하십시오.

명명된 구성

명명된 구성은 Application Server 등록 정보 설정을 캡슐화하는 추상입니다. 클러스터와 독립 실행형 서버 인스턴스는 명명된 구성을 참조하여 해당 등록 정보 설정을 가져옵니다. 명명된 구성을 갖는 J2EE 컨테이너 구성은 IP 주소, 포트 번호 및 힙 메모리 양과 같은 항목을 제외하고 컨테이너가 상주하는 물리적 시스템과 무관합니다. 명명된 구성을 사용하면 유연하고 강력한 Application Server 관리가 가능합니다.

명명된 구성의 등록 정보 설정을 변경하기만 하면 해당 구성을 참조하는 모든 클러스터와 독립 실행형 인스턴스에서 변경 사항을 수용하여 구성 변경 사항이 자동으로 적용됩니다. 명명된 구성은 해당 구성에 대한 모든 참조를 제거한 경우에만 삭제할 수 있습니다. 도메인은 여러 개의 명명된 구성을 포함할 수 있습니다.

Application Server에는 default-config라는 기본 구성이 함께 제공됩니다. 기본 구성은 Application Server Platform Edition의 경우 개발자 생산성이, Application Server Enterprise Edition의 경우 보안 및 고가용성이 최적화되어 있습니다.

기본 구성은 목적에 맞게 사용자 정의할 수 있으므로 이를 바탕으로 사용자만의 고유한 명명된 구성을 만들 수 있습니다. 명명된 구성은 관리 콘솔과 `asadmin` 명령줄 유틸리티를 사용하여 만들고 관리합니다.

HTTP 로드 밸런서 플러그인

로드 밸런서는 작업 로드를 여러 물리적 시스템 간에 분산시켜 시스템의 전체 처리량을 높입니다. Application Server Enterprise Edition에는 Sun Java System Web Server, Apache Web Server 및 Microsoft Internet Information Server용 로드 밸런서 플러그인이 포함되어 있습니다.

로드 밸런서 플러그인은 HTTP 및 HTTPS 요청을 받아들인 후 클러스터의 Application Server 인스턴스 중 하나로 전달합니다. 한 인스턴스에 오류가 발생하거나 네트워크 결함 때문에 사용할 수 없게 되거나 응답하지 않으면, 기존에 있던 사용 가능한 시스템으로 요청이 리디렉션됩니다. 또한 로드 밸런서는 실패한 인스턴스가 복구되었을 때를 인식하고 그에 따라 로드를 재분산할 수 있습니다.

상태 비보존형 경량 응용 프로그램의 경우 로드 균형 조정된 클러스터만으로 충분할 수 있습니다. 그러나 세션 상태를 포함하는 업무에 중요한 응용 프로그램의 경우 HADB와 함께 로드 균형 조정된 클러스터를 사용하십시오.

시스템에 로드 균형 조정을 설정하려면 Application Server뿐만 아니라 웹 서버 및 로드 밸런서 플러그인도 설치해야 하며 그 후에 다음을 수행해야 합니다.

- 로드 균형 조정에 참여할 Application Server 클러스터를 만듭니다.
- 응용 프로그램을 이 로드 균형 조정된 클러스터에 배포합니다.

로드 균형 조정에 참여하는 서버 인스턴스와 클러스터의 환경은 같습니다. 일반적으로 이것은 서버 인스턴스가 동일한 서버 구성을 참조하고, 동일한 물리적 자원에 액세스할 수 있으며, 서버 인스턴스에 동일한 응용 프로그램이 배포되어 있다는 것을 의미합니다. 동일한 환경은 작업 실패 전과 후에 로드 밸런서가 클러스터의 활성 인스턴스에 항상 로드를 균등하게 분산합니다.

`asadmin` 명령줄 도구를 사용하여 로드 밸런서 구성을 만들고, 클러스터와 서버 인스턴스에 대한 참조를 구성에 추가한 후 로드 밸런서가 참조할 클러스터를 활성화합니다. 그런 다음 로드 균형 조정할 응용 프로그램을 활성화하고, 상태 검사기를 만든 다음(선택 사항) 로드 밸런서 구성 파일을 생성하고, 마지막으로 로드 밸런서 구성 파일을 웹 서버 `config` 디렉토리로 복사합니다. 관리자는 스크립트를 만들어 전체 프로세스를 자동화할 수 있습니다.

자세한 내용과 전체 구성 지침을 보려면 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 5 장, “HTTP 로드 균형 조정 구성”을 참조하십시오.

세션 지속성

J2EE 응용 프로그램은 일반적으로 많은 양의 세션 상태 데이터를 포함하게 됩니다. 웹 장바구니가 세션 상태의 일반적인 예에 해당합니다. 또한 응용 프로그램은 자주 필요한 데이터를 세션 객체에 캐시할 수 있습니다. 실제로 사용자 상호 작용이 자주 발생하는 거의 모든 응용 프로그램에서는 세션 상태가 유지되어야 합니다. HTTP 세션과 SFSB(StateFul Session Bean)은 모두 세션 상태 데이터를 갖습니다.

세션 상태는 데이터베이스에 저장된 처리 상태만큼 중요하지는 않지만 최종 사용자에게는 서버 오류 전체에 대해 세션 상태를 보존하는 것이 중요할 수 있습니다.Application Server는 이 세션 상태를 저장소에 저장하거나 유지하는 기능을 제공합니다. 사용자 세션을 호스팅 중인 Application Server 인스턴스에 오류가 발생할 경우 세션 상태를 복구할 수 있습니다. 정보의 손실 없이 세션을 계속할 수 있습니다.

Application Server에서는 다음 유형의 세션 지속성 저장소가 지원됩니다.

- 메모리
- 고가용성(HA)
- 파일

메모리 지속성의 경우 항상 메모리에서 상태가 유지되고 오류 발생 후에는 지속되지 않습니다. HA 지속성의 경우 Application Server에서 HTTP 세션과 SFSB 세션에 대해 모두 HADB를 지속성 저장소로 사용합니다. 파일 지속성의 경우 Application Server에서 세션 객체를 일련화하여 세션 관리자 등록 정보에 지정된 파일 시스템 위치에 저장합니다. SFSB의 경우 HA가 지정되어 있지 않으면 Application Server에서 이 위치의 session-store 하위 디렉토리에 상태 정보를 저장합니다.

SFSB 상태에 저장해야 할 변경 사항이 있는지 검사하는 작업을 **검사점 지정**이라고 합니다. 활성화된 경우 트랜잭션이 롤백되는 경우에도 검사점 지정은 일반적으로 SFSB 관련 트랜잭션이 완료된 후에 수행됩니다. Stateful Session Bean 개발에 대한 자세한 내용은 **Sun Java System Application Server 9.1 Developer's Guide**의 "Using Session Beans"를 참조하십시오. SFSB 페일오버 활성화에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 "Stateful Session Bean 페일오버"를 참조하십시오.

세션 지속성 구성 설정은 Application Server에서 처리되는 요청의 수 외에도 HADB에서 수신하는 분당 요청 수뿐 아니라 각 요청의 세션 정보에 영향을 줍니다.

세션 지속성 구성에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 9 장, "고가용성 세션 지속성 및 페일오버 구성"을 참조하십시오.

클러스터의 IIOP 로드 균형 조정

IIOP 로드 균형 조정 기능을 사용하면 IIOP 클라이언트 요청이 여러 서버 인스턴스나 이름 서버에 분산됩니다. 목표는 클러스터 내에서 로드를 균일하게 분산시켜 확장성을 제공하는 것입니다. Sun Java System Application에서 EJB 클러스터링 및 가용성 기능과 결합된 IIOP 로드 균형 조정 기능은 로드 균형 조정뿐 아니라 EJB 페일오버 기능도 제공합니다.

클라이언트가 객체에 대해 JNDI 조회를 수행하면 이름 지정 서비스는 특정 서버 인스턴스와 연결된 `InitialContext`(IC) 객체를 만듭니다. 그러면 해당 IC 객체를 사용하여 수행된 모든 조회 요청은 동일한 서버 인스턴스로 보내집니다. 해당 `InitialContext`를 사용하여 조회된 모든 `EJBHome` 객체는 동일한 대상 서버에

호스트됩니다. 이후에 가져온 모든 Bean 참조 또한 동일한 대상 호스트에서 만들어집니다. 이 경우 `InitialContext` 객체를 만들 때 모든 클라이언트가 활성 대상 서버 목록을 임의화하므로 로드 균형 조정이 효과적으로 구현될 수 있습니다. 대상 서버 인스턴스가 작동 중단되면 조회 또는 EJB 메소드 호출은 다른 서버 인스턴스로 페일오버됩니다.

예를 들어 이 그림에 표시된 대로 ic1, ic2 및 ic3은 Client2의 코드에서 생성된 3개의 서로 다른 `InitialContext` 인스턴스로서, 클러스터 내 3개의 서버 인스턴스로 배포됩니다. 따라서, 이 클라이언트에서 만든 Enterprise JavaBeans가 3개의 인스턴스에 분산됩니다. Client1은 한 개의 InitialContext 객체만 생성했고 이 클라이언트의 Bean 참조는 Server Instance 1에만 있습니다. Server Instance 2가 다운되면 ic2의 조회 요청이 다른 서버 인스턴스(반드시 Server Instance 3일 필요는 없음)로 페일오버됩니다. 이전에 Server Instance 2에서 호스팅된 Bean에 대한 모든 Bean 메소드 호출 또한, 안전한 경우에 한해 다른 인스턴스로 자동으로 리디렉션됩니다. 조회 페일오버가 자동인 상태에서 Enterprise JavaBeans 모듈은 안전한 경우에 한해 메소드 호출을 다시 시도합니다.

IIOP 로드 균형 조정 및 페일오버는 투명하게 발생합니다. 응용 프로그램 배포 중에 특별한 단계가 필요하지는 않습니다. 클러스터에서 새 인스턴스를 추가 또는 삭제해도 클러스터의 기존 클라이언트 뷰가 업데이트되지 않습니다. 클라이언트측의 종점 목록을 수동으로 업데이트해야 합니다.

메시지 대기열 및 JMS 자원

Sun Java System Message Queue(MQ)는 분산 응용 프로그램에 대한 신뢰할 수 있는 비동기 메시징을 제공합니다. MQ는 JMS(Java Message Service) 표준을 구현하는 엔터프라이즈 메시징 시스템입니다. MQ는 MDB(Message-Driven Bean)와 같은 J2EE 응용 프로그램 구성 요소에 대한 메시징을 제공합니다.

Application Server는 Sun Java System Message Queue를 Application Server로 통합하여 JMS(Java Message Service) API를 구현합니다. Application Server Enterprise Edition에는 페일오버, 클러스터링 및 로드 균형 조정 기능이 있는 MQ Enterprise 버전이 포함되어 있습니다.

기본 JMS 관리 작업의 경우 Application Server 관리 콘솔과 `asadmin` 명령줄 유틸리티를 사용합니다.

Message Queue 클러스터 관리와 같은 고급 작업에는 *install_dir*`/imq/bin` 디렉토리에 제공된 도구를 사용합니다. Message Queue 관리에 대한 자세한 내용은 *Sun Java System Message Queue* **관리 설명서**를 참조하십시오.

메시지 페일오버를 위한 JMS 응용 프로그램 및 MQ 클러스터링 배포에 대한 자세한 내용은 52 페이지 “Message Queue 브로커 배포 계획”을 참조하십시오.

고가용성 데이터베이스

이 절은 다음 내용으로 구성되어 있습니다.

개요

J2EE 응용 프로그램의 세션 지속성에 대한 필요성은 이전에 26 페이지 “세션 지속성”에서 설명했습니다. Application Server는 가용성이 높은 세션 저장소로 HADB(고가용성 데이터베이스)를 사용합니다. HADB는 Application Server Enterprise Edition에 포함되어 있지만 별도의 호스트에 배포하여 실행할 수 있습니다. HADB는 HTTP 세션 및 Stateful Session Bean 데이터를 위한 가용성이 높은 데이터 저장소를 제공합니다.

이러한 분리된 구조의 이점은 다음과 같습니다.

- 고가용성 클러스터의 서버 인스턴스는 느슨하게 결합되고 고성능 J2EE 컨테이너 역할을 합니다.
- 서버 인스턴스의 중지 및 시작이 다른 서버 또는 이러한 서버의 가용성에 영향을 주지 않습니다.
- HADB를 다른 저비용 시스템 집합에서 실행할 수 있습니다(예: 단일 또는 이중 프로세서 사용). 여러 클러스터가 이러한 시스템을 공유할 수 있습니다. 배포 요구 사항에 따라 Application Server와 동일한 시스템(공존)이나 다른 시스템(개별 계층)에서 HADB를 실행할 수 있습니다. 두 가지 옵션에 대한 자세한 내용은 61 페이지 “공존 토폴로지”를 참조하십시오.
- 상태 관리 요구 사항이 변경되면 기존 클러스터나 클러스터의 응용 프로그램에 영향을 주지 않고 HADB 시스템에 자원을 추가할 수 있습니다.

주 – HADB는 Application Server용으로 최적화된 데이터베이스로서, 응용 프로그램에서 일반 데이터베이스로 사용하도록 개발되지 않았습니다.

HADB 하드웨어 및 네트워크 시스템 요구 사항을 보려면 **Sun Java System Application Server 9.1 릴리스 노트**의 “하드웨어 및 소프트웨어 요구 사항”을 참조하십시오. HADB에 필요한 추가 시스템 구성 단계를 보려면 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 2 장, “고가용성 데이터베이스 설치 및 설정”을 참조하십시오.

시스템 요구 사항

HADB 호스트에 대한 시스템 요구 사항은 다음과 같습니다.

- HADB 노드당 CPU 1개 이상
- 노드당 메모리 512MB 이상

네트워크 구성 요구 사항을 보려면 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 2 장, "고가용성 데이터베이스 설치 및 설정"을 참조하십시오. 고가용성을 위한 추가 요구 사항을 보려면 33 페이지 "이중 오류 완화"를 참조하십시오.

HADB 구조

HADB는 **노드** 쌍으로 구성된 분산 시스템입니다. 노드는 두 개의 DRU(Data Redundancy Units)로 구분되는데, 31 페이지 "DRU(Data Redundancy Units)"의 설명처럼 각 DRU에는 각 노드 쌍의 한 노드가 포함됩니다.

각 노드를 구성하는 요소는 다음과 같습니다.

- 트랜잭션 상태 복제를 위한 프로세스 모음
- 프로세스 간 통신에 사용되는 전용 공유 메모리 영역
- 보조 저장 장치(디스크) 1개 이상

한 개의 HADB 노드 세트는 하나 이상의 **세션 데이터베이스**를 호스팅할 수 있습니다. 세션 데이터베이스는 각각 다른 Application Server 클러스터와 연결됩니다. 클러스터를 삭제하면 연결된 세션 데이터베이스도 삭제됩니다.

HADB 하드웨어 요구 사항을 보려면 **Sun Java System Application Server 9.1 릴리스 노트**의 "하드웨어 및 소프트웨어 요구 사항"을 참조하십시오.

노드 및 노드 프로세스

HADB 노드에는 두 가지 유형이 있습니다.

- 데이터를 저장하는 **활성 노드**
- 처음에는 데이터를 포함하지 않지만 활성 노드가 사용할 수 없는 상태가 되었을 때 활성 노드로 작동하는 **예비 노드**예비 노드는 선택 사항이지만 고가용성을 향상시키려는 경우 유용합니다.

각 노드에는 한 개의 부모 프로세스와 여러 개의 자식 프로세스가 있습니다. NSUP(노드 수퍼바이저)라고 하는 부모 프로세스는 관리 에이전트에 의해 시작되며, 자식 프로세스를 만들어 실행 상태를 유지하는 작업을 수행합니다.

자식 프로세스는 다음과 같습니다.

- 분산 노드에서 트랜잭션를 조정하고 데이터 저장소를 관리하는 트랜잭션 서버 프로세스(TRANS)

- 정렬 및 결합과 같은 복합 관계 대수 쿼리를 조정 및 실행하는 관계 대수 서버 프로세스(RELALG)
- SQL 사전 캐시를 유지하는 SQL 공유 메모리 서버 프로세스(SQLSHM)
- 클라이언트 쿼리를 수신하여 로컬 HADB 명령으로 컴파일한 후 TRANS로 전송하고, 결과를 수신하여 클라이언트로 전달하는 SQL 서버 프로세스(SQLC). 각 노드에는 클라이언트 연결별로 주 SQL 서버 한 대와 하위 서버 한 대가 있습니다.
- 관리 에이전트가 hadbm 관리 클라이언트에서 지시한 관리 명령을 실행하기 위해 사용하는 노드 관리자 서버 프로세스(NOMAN)

DRU(Data Redundancy Units)

앞에서 설명한 대로 HADB 인스턴스에는 한 쌍의 DRU가 포함되어 있습니다. 각 DRU는 쌍을 이루는 다른 DRU와 동일한 수의 활성 노드와 예비 노드를 갖습니다. 한 DRU의 각 활성 노드에는 다른 DRU의 **미러 노드**가 있습니다. 각 DRU에는 미러링으로 인해 전체 데이터베이스의 복사본이 포함되어 있습니다.

다음 그림은 활성 노드 4개와 예비 노드 두 개로 구성된 6개의 노드가 있는 HADB 구조 예를 보여줍니다. 노드 0과 노드 1은 하나의 미러 쌍이고 노드 2와 노드 3도 미러 쌍입니다. 이 예에서 각 호스트에는 하나의 노드가 있습니다. 일반적으로, 시스템 자원이 충분할 경우 호스트에 둘 이상의 노드가 있을 수 있습니다(30 페이지 “시스템 요구 사항” 참조).

주 – 각 DRU에 한 시스템씩 HADB 노드를 쌍으로 호스팅하는 시스템을 추가해야 합니다.

HADB는 데이터와 서비스를 복제하여 고가용성을 달성합니다. 미러 노드의 데이터 복제본은 **기본 복제본**과 **상시 대기 복제본**으로 지정됩니다. 기본 복제본은 삽입, 삭제, 업데이트 및 읽기와 같은 작업을 수행합니다. 상시 대기 복제본은 기본 복제본 작업의 로그 레코드를 수신하여 트랜잭션 수명 내에서 재실행합니다. 읽기 작업은 기본 노드에 의해서만 수행되므로 기록되지 않습니다. 각 노드는 기본 복제본과 상시 대기 복제본을 모두 포함하고 두 역할을 모두 수행합니다. 데이터베이스는 단편화되어 DRU의 활성 노드에 분산됩니다. 미러 쌍의 각 노드는 동일한 데이터 단편 집합을 포함합니다. 미러 노드의 데이터 중복화를 **복제**라고 합니다. 복제를 사용하면 HADB에서 고가용성을 제공할 수 있습니다. 노드에 오류가 발생하면 미러 노드가 몇 초 내로 거의 즉시 인계를 받습니다. 복제는 가용성을 확보하고 데이터나 서비스 손실 없이 노드 오류 또는 DRU 오류를 마스크합니다.

오류 발생 노드의 기능을 인계 받은 미러 노드는 자체 작업과 함께 오류 발생 노드의 작업까지 이중 작업을 수행해야 합니다. 미러 노드의 자원이 부족한 경우 오버로드로 인해 성능이 저하되고 오류 가능성이 높아집니다. 노드에 오류가 발생하면 HADB는 노드를 다시 시작하려고 시도합니다. 하드웨어 장애 등으로 인해 오류 발생 노드가 다시 시작되지 않으면 시스템은 계속 작동하지만 가용성이 감소합니다.

HADB는 노드 하나, 전체 DRU 또는 여러 노드의 오류를 허용하지만 노드와 해당 미러에 모두 오류가 발생하는 "이중 오류"는 허용하지 않습니다. 이중 오류 가능성을 줄이는 방법에 대한 자세한 내용은 33 페이지 “이중 오류 완화”를 참조하십시오.

예비 노드

노드에 오류가 발생하면 해당 미러 노드가 작업을 인계 받습니다. 오류 노드에 예비 노드가 없으면 이 때 오류 노드에 미러가 없습니다. 예비 노드는 오류 발생 노드의 미러를 자동으로 복제합니다. 예비 노드가 있으면 시스템이 미러 노드 없이 작동하는 시간이 감소합니다.

예비 노드는 일반적으로 데이터를 포함하지 않지만 DRU의 활성 노드에 오류가 있는지 지속적으로 모니터합니다. 한 노드에 오류가 발생하고 지정된 시간 초과 기간 내에 복구되지 않으면 예비 노드가 미러 노드에서 데이터를 복사하여 동기화합니다. 이 작업에 소요되는 시간은 복사되는 데이터 양과 시스템 및 네트워크 용량에 따라 달라집니다. 동기화한 후 예비 노드가 수동 개입 없이 미러 노드를 자동으로 대체하므로 미러 노드가 오버로드에서 해제되어 미러의 로드 균형이 조정됩니다. 이 작업은 **페일백** 또는 **자체 복구**라고도 합니다.

하드웨어를 바꾸거나 소프트웨어를 업그레이드하여 오류 발생 호스트가 복구 및 다시 시작되면 원래 예비 노드가 현재 활성 상태이므로 호스트에서 실행 중인 노드가 예비 노드로 시스템에 참가합니다.

예비 노드가 필수 항목은 아니지만 이를 사용하여 시스템에 오류가 발생한 경우에도 시스템의 전체 서비스 수준을 유지할 수 있습니다. 또한 예비 노드를 사용하면 활성 노드를 호스팅하는 시스템에서 계획된 유지 보수를 손쉽게 수행할 수 있습니다. 시스템 중 하나에 오류가 발생할 경우 HADB 시스템이 성능과 가용성에 악영향을 주지 않고 계속 작동하도록 DRU에 예비 시스템 역할을 수행할 시스템을 각각 하나씩 할당합니다.

주 – 일반적으로 사용할 수 없게 되는 시스템을 대체할 만큼 Application Server 인스턴스 및 HADB 노드가 충분한 예비 시스템을 사용합니다.

예비 노드 구성 예

다음 예는 HADB 배포에서 예비 노드의 사용을 보여줍니다. 가능한 두 가지 배포 토폴로지로 HADB 및 Application Server가 동일한 호스트에 상주하는 **공존** 토폴로지와 각각 다른 호스트에 상주하는 **개별 계층** 토폴로지가 있습니다. 배포 토폴로지에 대한 자세한 내용은 3 장을 참조하십시오.

예: 공존 구성

예비 노드 구성 예로서 각 서버에 Application Server 인스턴스 하나와 HADB 데이터 노드 둘이 있는 4대의 Sun Fire™ V480 서버로 구성된 공존 토폴로지가 있다고 가정합니다.

예비 노드로 두 대의 서버를 추가로 할당합니다(DRU당 시스템 한 대). 각 예비 시스템은 Application Server 인스턴스 하나와 예비 HADB 노드 둘을 실행합니다.

예: 개별 계층 구성

HADB 계층에 서버당 두 개의 HADB 데이터 노드를 실행하는 두 대의 Sun Fire™ 280R 서버가 있는 개별 계층 토폴로지가 있다고 가정합니다. 이 시스템에서 시스템 한 대가 사용할 수 없게 된 경우에도 성능을 유지하려면 Application Server 인스턴스 계층과 HADB 계층에 사용할 예비 시스템을 각각 하나씩 구성합니다.

Application Server 인스턴스 계층용 예비 시스템에는 Application Server 인스턴스 계층의 다른 시스템과 동일한 수의 인스턴스가 있어야 합니다. 마찬가지로 HADB 계층용 예비 시스템에도 HADB 계층의 다른 시스템과 동일한 수의 HADB 노드가 있어야 합니다.

이중 오류 완화

HADB의 기본 제공 데이터 복제를 사용하여 단일 노드 또는 전체 DRU의 오류를 허용하도록 할 수 있습니다. 기본적으로 HADB는 미러 노드 쌍이나 양쪽 DRU에 모두 오류가 발생하는 **이중 오류**를 허용하지 않습니다. 이 경우 HADB를 사용할 수 없게 됩니다.

이전 절의 설명대로 예비 노드를 사용하고 다음 단계를 수행하면 이중 오류 발생 가능성을 최소화할 수 있습니다.

- **독립 전원 공급 장치 제공**: 내결함성을 최적화하려면 하나의 DRU를 지원하는 서버에 무정전 전원 공급 장치를 통한 독립 전원, 처리 장치 및 저장소가 있어야 합니다. 한 쪽 DRU에서 전원 오류가 발생할 경우 전원이 회복될 때까지 다른 DRU의 노드에서 요청이 처리됩니다.
- **이중 상호 연결 제공**: 단일 네트워크 오류를 허용하려면 DRU 간에 라인과 스위치를 복제합니다.

이 단계는 선택 사항이지만 HADB 인스턴스의 전반적인 가용성을 높여줍니다.

HADB 관리 시스템

HADB 관리 시스템은 기본 제공 보안 기능이 포함되어 있으며 다중 플랫폼 관리를 용이하게 합니다. 아래의 그림에서 보여주는 것처럼 HADB 관리 구조는 다음 구성 요소로 구성됩니다.

- 34 페이지 “관리 클라이언트”
- 34 페이지 “관리 에이전트”
- 35 페이지 “관리 도메인”
- 35 페이지 “저장소”

그림과 같이 HADB 서비스를 실행하는 모든 시스템에서는 HADB 관리 에이전트가 실행됩니다. 각 시스템은 대개 하나 이상의 HADB 노드를 호스팅합니다. HADB 관리 도메인에는 Application Server 도메인처럼 여러 시스템이 포함되어 있습니다.

데이터베이스가 내결함성을 갖도록 구성하려면 도메인에 두 대 이상의 시스템이 필요하고 일반적으로 DRU 쌍을 형성할 수 있도록 시스템이 짝수로 존재해야 합니다. 따라서 도메인에 많은 관리 에이전트가 포함됩니다.

그림과 같이 도메인은 하나 이상의 데이터베이스 인스턴스를 포함할 수 있습니다. 시스템 한 대에는 하나 이상의 데이터베이스 인스턴스에 속한 하나 이상의 노드가 포함될 수 있습니다.

관리 클라이언트

HADB **관리 클라이언트**는 HADB 도메인과 해당 데이터베이스 인스턴스를 관리하기 위한 명령줄 유틸리티 `hadbm`입니다. 연결된 Application Server 클러스터가 중지된 경우에도 HADB 서비스는 계속 실행될 수 있지만 삭제하려는 경우에는 조심스럽게 종료해야 합니다. `hadbm` 사용에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 3 장, "고가용성 데이터베이스 관리"를 참조하십시오.

asadmin 명령줄 유틸리티를 사용하여 가용성이 높은 클러스터와 연결된 HADB 인스턴스를 만들거나 삭제할 수 있습니다. 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 9 장, "고가용성 세션 지속성 및 페일오버 구성"을 참조하십시오.

관리 에이전트

관리 에이전트는 호스트의 자원에 액세스할 수 있는 `ma`라는 서버 프로세스로서, 장치를 만들거나 데이터베이스 프로세스를 시작하는 등의 작업을 수행할 수 있습니다. 관리 에이전트는 데이터베이스 인스턴스 시작 또는 중지와 같은 관리 클라이언트 명령을 조정 및 수행합니다.

관리 클라이언트는 에이전트의 주소와 포트 번호를 지정하여 관리 에이전트에 연결합니다. 연결된 후 관리 클라이언트는 관리 에이전트를 통해 HADB에 명령을 보내고 에이전트는 요청을 수신하여 실행합니다. 따라서 관리 에이전트는 `hadbm` 관리 명령을 호스트에 지시하기 전에 해당 호스트에서 실행 중이어야 합니다. 관리 에이전트는 자동으로 시작되는 시스템 서비스로 구성할 수 있습니다.

관리 에이전트 가용성 보장

관리 에이전트 프로세스는 HADB 노드 수퍼바이저 프로세스가 실패한 경우 프로세스를 다시 시작하여 HADB의 가용성을 보장합니다. 따라서 배포 시 HADB의 전체적인 가용성을 유지하려면 `ma` 프로세스의 가용성을 보장해야 합니다. 다시 시작한 후 관리 에이전트는 도메인의 다른 에이전트에서 도메인 및 데이터베이스 구성 데이터를 복구합니다.

호스트 운영 체제(OS)를 사용하여 관리 에이전트의 가용성을 보장합니다. Solaris나 Linux의 경우 `init.d`를 사용하여 프로세스 오류와 운영 체제 재부트 이후 `ma` 프로세스의 가용성을 보장할 수 있습니다. Windows의 경우 관리 에이전트가 Windows 서비스로 실행됩니다. 따라서 에이전트에 오류가 발생하거나 OS가 재부트될 경우 OS에서 관리 에이전트를 다시 시작합니다.

관리 도메인

HADB 관리 도메인은 각기 동일한 포트 번호에서 실행되는 관리 에이전트가 있는 호스트의 집합입니다. 도메인의 호스트는 하나 이상의 HADB 데이터베이스 인스턴스를 포함할 수 있습니다. 관리 도메인은 에이전트가 사용하는 공통 포트 번호 및 도메인을 만들거나 도메인에 에이전트를 추가할 때 생성되는 **도메인 키**라는 식별자로 정의됩니다. 도메인 키는 도메인의 고유 식별자로서 관리 에이전트가 멀티캐스트를 사용하여 통신하기 때문에 중요한 역할을 합니다. Application Server 도메인과 일치하는 HADB 관리 도메인을 설정할 수 있습니다.

한 도메인에 데이터베이스 인스턴스가 여러 개 있으면 여러 개발자 그룹에서 각각 전용 데이터베이스 인스턴스를 사용할 수 있기 때문에 개발 환경에서 유용할 수 있으며 프로덕션 환경에서도 경우에 따라 유용할 수 있습니다.

도메인에 속한 모든 에이전트는 해당 관리 작업을 조정합니다. `hadbm` 명령을 통해 데이터베이스 구성을 변경하면 모든 에이전트가 이에 따라 구성을 변경합니다. 노드의 호스트에서 관리 에이전트가 실행되고 있지 않으면 해당 노드를 중지하거나 다시 시작할 수 없습니다. 그러나 일부 에이전트를 사용할 수 없는 경우에도 HADB 상태 또는 구성 변수 값을 읽는 `hadbm` 명령을 실행할 수 있습니다.

다음 관리 클라이언트 명령을 사용하여 관리 도메인 작업을 수행합니다.

- **hadbm createdomain**: 지정된 호스트를 사용하여 관리 도메인을 만듭니다.
- **hadbm extenddomain**: 호스트를 기존 관리 도메인에 추가합니다.
- **hadbm deletedomain**: 관리 도메인을 제거합니다.
- **hadbm reducedomain**: 관리 도메인에서 호스트를 제거합니다.
- **hadbm listdomain**: 관리 도메인에 정의된 모든 호스트를 나열합니다.

저장소

관리 에이전트는 데이터베이스 구성을 **저장소**에 저장합니다. 저장소는 모든 관리 에이전트에서 복제되기 때문에 내결함성이 우수합니다. 서버에서 구성을 유지하면 관리 클라이언트가 설치된 모든 컴퓨터에서 관리 작업을 수행할 수 있습니다.

저장소 내용을 변경하려면 도메인에 속한 대부분의 관리 에이전트가 실행 중이어야 합니다. 따라서 도메인에 M개의 에이전트가 있는 경우 저장소 내용을 변경하려면 에이전트가 M/2+1개(정수로 절사) 이상 실행 중이어야 합니다.

하드웨어 장애 등으로 인해 도메인의 일부 호스트를 사용할 수 없는 상태에서 쿼럼이 없기 때문에 일부 관리 명령을 수행할 수 없는 경우 `hadbm disablehost` 명령을 사용하여 도메인에서 오류 발생 호스트를 제거합니다.

설정 및 구성 로드맵

▼ 고가용성 구현을 위한 Application Server의 설정 및 구성 방법

1 **1 장의 설명대로 성능과 QoS 요구 사항 및 목적을 결정합니다.**

2 **1 장의 49 페이지 "설계 결정 사항"에 설명된 대로 시스템 크기를 설정합니다.**

- Application Server 인스턴스 수
- HADB 노드 및 호스트 수
- HADB 저장소 용량

3 **3 장의 설명대로 시스템 토폴로지를 결정합니다.**

HADB를 Application Server와 동일한 호스트 시스템에 설치할 것인지 다른 시스템에 설치할 것인지를 결정하는 단계입니다.

4 **HADB, 웹 서버 등 관련된 하위 구성 요소와 함께 Application Server 인스턴스를 설치합니다.**

5 **도메인과 클러스터를 만듭니다.**

6 **웹 서버 소프트웨어를 구성합니다.**

7 **로드 밸런서 플러그인을 설치합니다.**

8 **로드 균형 조정 기능을 설정 및 구성합니다.**

9 **HADB 노드와 DRU를 설정 및 구성합니다.**

10 **HA 세션 지속성을 사용하도록 AS 웹 컨테이너와 EJB 컨테이너를 구성합니다.**

11 **응용 프로그램을 배포하고 고가용성과 세션 페일오버를 사용하도록 구성합니다.**

12 **광범위하게 메시징을 사용하는 경우 페일오버를 사용하도록 JMS 클러스터를 구성합니다.**

자세한 내용은 *Sun Java System Message Queue* **관리 설명서**를 참조하십시오.

◆ ◆ ◆ 2 장

2 배포 계획

Application Server를 배포하기 전에 먼저 성능 및 가용성 목표를 결정한 후 이에 따라 하드웨어, 네트워크 및 저장소 요구 사항을 결정해야 합니다.

이 장은 다음 내용으로 구성되어 있습니다.

성능 목표 설정

가장 단순한 의미에서 높은 성능이란 처리량 최대화와 응답 시간 감소를 의미합니다. 하지만 이러한 기본 목표를 넘어, 다음을 결정하여 특정 목표를 설정할 수 있습니다.

- 배포되는 응용 프로그램 및 서비스 유형과 클라이언트에서 액세스하는 방법
- 높은 가용성이 요구되는 응용 프로그램 및 서비스
- 응용 프로그램이 세션 상태를 포함하는지 여부
- 시스템에서 지원해야 하는 요청 용량 또는 처리량
- 시스템에서 지원해야 하는 동시 사용자 수
- 사용자 요청에 대해 허용되는 평균 응답 시간
- 요청 간 평균 인지 시간

이러한 메트릭 중 일부는 RBE(원격 브라우저 에뮬레이터) 도구나 예상 응용 프로그램 활동을 시뮬레이트하는 웹 사이트 성능 및 벤치마킹 소프트웨어를 사용하여 계산할 수 있습니다. 일반적으로 RBE와 벤치마킹 제품은 동시 HTTP 요청을 생성한 후 지정된 분당 요청 수에 대한 응답 시간을 보고합니다. 이렇게 산출된 수치를 사용하여 서버 활동을 계산할 수 있습니다.

이 장에 설명된 계산 결과는 절대적 결과가 아닙니다. Application Server와 응용 프로그램의 성능을 미세 조정할 때 기준으로 적용할 참조점으로 사용하십시오.

이 절은 다음 내용으로 구성되어 있습니다.

처리량 예측

넓은 의미에서 **처리량**이란 Application Server에서 수행되는 작업의 양을 나타냅니다. Application Server의 경우 처리량은 서버 인스턴스별 분당 처리되는 요청 수로 정의할 수 있습니다. 고가용성 응용 프로그램은 세션 상태 데이터를 정기적으로 저장하기 때문에 HADB에도 일정 수준 이상의 처리량을 요구합니다. HADB의 처리량은 분당 HADB 요청 수와 요청당 평균 세션 크기를 곱한 값인 분당 저장된 세션 데이터의 양으로 정의할 수 있습니다.

다음 절에 설명된 것처럼 Application Server의 처리량은 사용자 요청의 특징과 크기, 사용자 수, 그리고 Application Server 인스턴스와 백엔드 데이터베이스의 성능을 비롯한 많은 요소로 구성된 함수입니다. 시뮬레이트된 작업 로드를 가지고 벤치마킹함으로써 단일 시스템의 처리량을 예측할 수 있습니다.

고가용성 응용 프로그램은 데이터를 HADB에 정기적으로 저장하기 때문에 추가 오버헤드가 발생합니다. 오버헤드 크기는 데이터의 양, 변경 빈도 및 저장 빈도에 따라 결정됩니다. 처음 두 요소는 해당 응용 프로그램에 따라 결정되며 마지막 요소도 서버 설정의 영향을 받습니다.

HADB 처리량은 분당 HADB 요청 수에 요청당 평균 데이터 양을 곱한 값으로 정의할 수 있습니다. HADB 처리량이 클수록 더 많은 HADB 노드와 더 큰 저장소가 필요합니다.

Application Server 인스턴스의 로드 예측

다음 요소를 고려하여 Application Server 인스턴스의 로드를 예측합니다.

최대 동시 사용자 수

사용자는 웹 브라우저나 Java 프로그램과 같은 클라이언트를 통해 응용 프로그램과 상호 작용합니다. 클라이언트는 사용자의 작업에 따라 정기적으로 Application Server에 요청을 보냅니다. 세션이 만료되거나 종료되지 않는 한 사용자는 **활성** 상태로 간주됩니다. 동시 사용자 수를 예측할 때 이러한 모든 활성 사용자를 포함합니다.

다음 그림은 분당 처리되는 요청 수(처리량) 대 사용자 수를 나타내는 일반적인 그래프입니다. 처음에는 사용자 수가 증가함에 따라 처리량이 증가합니다. 그러나 동시 요청 수가 증가함에 따라 서버 성능이 포화되기 시작하고 처리량이 감소하기 시작합니다.

동시 사용자를 추가함으로써 분당 처리할 수 있는 요청 수가 감소하는 지점을 확인할 수 있습니다. 이 지점은 최적 성능에 도달한 후 처리량이 감소하기 시작하는 시점을 나타냅니다. 일반적으로, 가능한 한 높은 최적 처리량으로 시스템을 작동하도록 노력해야 합니다. 추가 로드를 처리하고 처리량을 높이기 위해 처리 용량을 추가해야 할 수 있습니다.

인지 시간

사용자는 요청을 지속적으로 제출하지 않습니다. 사용자가 요청을 제출하면 서버에서는 요청을 수신하여 처리한 후 사용자가 새 요청을 제출하기 전 시간이 소비되는 시점에 결과를 반환합니다. 한 요청과 다음 요청 사이의 시간을 **인지 시간**이라고 합니다.

인지 시간은 사용자 유형에 따라 다릅니다. 예를 들어 웹 서비스의 경우와 같은 시스템 간 상호 작용은 일반적으로 사람이 사용자인 경우보다 낮은 인지 시간을 포함합니다. 또한, 시스템과 인간 상호 작용의 혼합을 고려하여 인지 시간을 예측해야 할 수 있습니다.

평균 인지 시간 결정은 중요한 작업입니다. 이 기간을 사용하여 분당 완료해야 하는 요청 수는 물론 시스템에서 지원할 수 있는 동시 사용자 수를 계산할 수 있습니다.

평균 응답 시간

응답 시간이란 Application Server가 사용자에게 요청 결과를 반환하는 데 걸리는 시간을 말합니다. 응답 시간은 네트워크 대역폭, 사용자 수, 제출된 요청의 수와 유형, 그리고 평균 인지 시간과 같은 요소의 영향을 받습니다.

이 절에서 응답 시간은 중간 또는 평균 응답 시간을 나타냅니다. 각 요청 유형에는 고유한 최소 응답 시간이 있습니다. 그러나 시스템 성능을 평가할 때는 모든 요청의 평균 응답 시간을 기반으로 분석합니다.

응답 시간이 빠를수록 처리되는 분당 요청 수는 증가합니다. 하지만 다음 그림에 나타난 것처럼 분당 요청 수가 감소하더라도 시스템의 사용자 수가 증가함에 따라 응답 시간이 증가하기 시작합니다.

죄송합니다. 이 설명서 버전에서는 그래픽을 사용할 수 없습니다.

이 그림과 같은 시스템 성능 그래프에 따르면 특정 시점 이후 분당 요청 수는 응답 시간에 반비례합니다. 분당 요청 감소량이 급격할수록 응답 시간의 증가가 더 가파릅니다(점선 화살표로 표시).

그림에서 최대 로드 지점은 분당 요청 수가 감소하기 시작하는 지점입니다. 이 지점 이전의 응답 시간 계산에는 공식에 최대 수치가 사용되지 않기 때문에 반드시 정확할

필요는 없습니다. 이 지점 이후에는 분당 요청과 응답 시간 사이의 반비례 관계 때문에 관리자가 최대 사용자 수 및 분당 요청 수를 사용하여 응답 시간을 더욱 정확하게 계산할 수 있습니다.

다음 공식을 사용하여 최대 로드 시 응답 시간(초)인 $T_{response}$를 결정합니다.

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

여기서,

- n은 동시 사용자 수입니다.
- r은 서버가 수신하는 초당 요청 수입니다.
- T_{think}는 평균 인지 시간(초)입니다.
  정확한 응답 시간 결과를 얻기 위해서는 항상 수식에 인지 시간을 포함해야 합니다.

예 2–1 응답 시간 계산

다음과 같은 조건이 있는 경우

- 최대 로드 시 시스템에서 지원할 수 있는 최대 동시 사용자 수 n은 5,000입니다.
- 최대 로드 시 시스템에서 처리할 수 있는 최대 요청 수 r은 초당 1,000입니다.

평균 인지 시간 T_{think}는 요청당 3초입니다.

즉, 응답 시간은 다음과 같이 계산됩니다.

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

따라서 응답 시간은 2초입니다.

특히 최대 로드 시 시스템 응답 시간을 계산한 후 응용 프로그램에 대한 허용되는 응답 시간과 비교합니다. 처리량과 함께 응답 시간은 Application Server의 성능에 있어 중요한 기본 요소 중 한 가지입니다.

분당 요청 수

지정된 시간의 동시 사용자 수, 해당 요청의 응답 시간 및 평균 사용자 인지 시간을 알고 있으면 분당 요청 수를 계산할 수 있으며, 일반적으로 시스템에 있는 동시 사용자 수 예측부터 시작합니다.

예를 들어 웹 사이트 성능 소프트웨어를 실행한 후 관리자는 온라인 은행 웹 사이트에서 요청을 제출하는 평균 동시 사용자 수가 3,000이라는 결론을 내립니다. 이 수치는 온라인 은행 회원으로 가입한 사용자 수, 사용자의 은행 업무 트랜잭션 동작, 사용자가 요청을 제출하려고 선택한 시간대나 요일 등에 따라 결정됩니다.

따라서 이 정보를 알고 있으면 이 절에 설명된 분당 요청 수에 대한 공식을 사용하여 시스템에서 이 사용자 기준에 따라 처리할 수 있는 분당 요청 수를 계산할 수 있습니다.

최대 로드 시 분당 요청 수와 응답 시간은 반비례하므로 응답 시간 향상을 위해 허용되는 분당 요청 수를 낮출 것인지 분당 요청 수를 높이기 위해 허용되는 응답 시간을 늦출 것인지를 결정합니다.

시스템 성능을 미세 조정할 시작점에서 허용되는 분당 요청 수 및 응답 시간 임계값을 시험합니다. 그런 다음 조정이 필요한 시스템 영역을 결정합니다.

이전 절의 수식에서 r은 다음과 같이 구할 수 있습니다.

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

예 2–2 분당 요청 수 계산

조건은 다음과 같습니다.

- $n = 2{,}800$ 동시 사용자
- $T_{response} = 1$(요청당 평균 응답 시간 1초)
- $T_{think} = 3$(평균 인지 시간 3초)

초당 요청 수는 다음과 같이 계산합니다.

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

따라서 초당 요청 수는 700이고 분당 요청 수는 42000입니다.

HADB의 로드 예측

HADB의 로드 계산에는 다음 요소를 고려합니다.

- 41 페이지 “HTTP 세션 지속성 빈도”
- 42 페이지 “HTTP 세션 크기 및 범위”
- 43 페이지 “Stateful Session Bean 검사점 지정”

세션 지속성 구성에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 9 장, “고가용성 세션 지속성 및 페일오버 구성”을 참조하십시오.

HTTP 세션 지속성 빈도

HADB에서 수신하는 분당 요청 수는 **지속성 빈도**에 따라 결정됩니다. 지속성 빈도는 Application Server가 HTTP 세션 데이터를 HADB에 저장하는 빈도를 결정합니다.

지속성 빈도 옵션은 다음과 같습니다.

- **web-method**(기본값): 서버가 모든 HTTP 응답과 함께 세션 데이터를 저장합니다. 이 옵션을 사용하면 저장된 세션 정보가 최신 상태를 유지하지만 HADB에 트래픽이 높아집니다.
- **time-based**: 세션이 지정된 시간 간격으로 저장됩니다. 이 옵션을 사용하면 HADB에 대한 트래픽이 감소하지만 세션 정보가 최신 상태를 보장할 수 없습니다.

다음 표는 지속성 빈도 옵션의 장점 및 단점에 대한 간략한 설명입니다.

표 2–1 지속성 빈도 옵션 비교

지속성 빈도 옵션	장점	단점
web-method	항상 최신 세션 정보를 사용할 수 있습니다.	응답 시간이 증가하고 처리량이 감소할 수 있습니다.
time-based	응답 시간이 향상되고 처리량이 향상될 수 있습니다.	Application Server 인스턴스에 오류가 발생한 뒤에 최신 세션 정보를 사용하지 못할 수 있습니다.

HTTP 세션 크기 및 범위

요청당 세션 크기는 세션에 저장된 세션 정보의 양에 따라 결정됩니다.

정보 – 전체 성능을 향상시키려면 세션 정보의 양을 최대한 줄이십시오.

지속성 범위 설정을 통해 요청당 세션 크기를 미세 조정할 수 있습니다. 다음 HTTP 세션 지속성 범위 옵션 중에서 선택합니다.

- **session**: 서버에서 세션 정보를 HADB에 저장할 때마다 전체 세션 객체를 일련화하여 저장합니다.
- **modified-session**: 세션이 수정된 경우에만 서버에서 세션을 저장합니다. Bean의 `setAttribute()` 메소드에 대한 호출을 가로채 수정을 감지합니다. 이 옵션은 내부 객체에 대한 직접 수정을 감지하지 않으므로 이 경우 `setAttribute()`를 명시적으로 호출하도록 SFSB를 코딩해야 합니다.
- **modified-attribute**: 서버에서 세션이 마지막으로 저장된 이후 수정(삽입, 업데이트 또는 삭제)된 속성만 저장합니다. modified-session과 동일한 단점을 갖지만 제대로 적용될 경우 HADB 쓰기 처리량 요구 사항을 크게 줄일 수 있습니다.

이 옵션을 사용하려면 응용 프로그램이 다음을 충족해야 합니다.

- 세션 상태를 수정할 때마다 `setAttribute()` 또는 `removeAttribute()`를 호출합니다.
- 속성 간에는 상호 참조가 없어야 합니다.
- 여러 속성에서 세션 상태를 분배하거나 최소한 읽기 전용 속성 및 수정 가능한 속성 간에 세션 상태를 분배합니다.

  다음 표는 지속성 범위 옵션의 장점 및 단점에 대한 간략한 설명입니다.

표 2–2 지속성 범위 옵션 비교

지속성 범위 옵션	장점	단점
modified-session	세션 상태를 수정하지 않고 요청에 대한 향상된 응답 시간을 제공합니다.	일반적으로 웹 메소드 `doGet()` 또는 `doPost()`를 실행하는 동안 응용 프로그램은 다음 세션 메소드를 호출해야 합니다. ■ 속성이 변경된 경우 `setAttribute()` ■ 속성이 제거된 경우 `removeAttribute()`
session	응용 프로그램에 대한 제약 조건이 없습니다.	`modified-session` 및 `modified-attribute` 옵션에 비해 처리량과 응답 시간이 저하될 수 있습니다.
modified-attribute	수정된 세션 상태 비율이 낮은 요청에 대한 처리량과 응답 시간이 향상됩니다.	지정된 요청에 대해 수정된 세션 상태 비율이 60%에 가까우면 처리량과 응답 시간이 저하됩니다. 이 경우 속성을 개별 레코드로 분할하는 오버헤드 때문에 성능이 다른 옵션보다 더 저하됩니다.

Stateful Session Bean 검사점 지정

SFSB 세션 지속성의 경우 HADB의 로드는 다음에 따라 결정됩니다.

- 검사점 지정을 위해 활성화된 SFSB 수
- 검사점 지정을 위해 선택된 SFSB 메소드 및 사용 빈도
- 세션 객체 크기
- 트랜잭션 메소드

트랜잭션이 롤백되는 경우에도 검사점 지정은 일반적으로 SFSB 관련 트랜잭션이 완료된 후에 수행됩니다.

성능을 향상시키려면 검사점 지정을 위해 소수의 메소드 집합을 지정합니다. 검사점 지정되는 데이터 크기와 검사점 지정 빈도에 따라 지정된 클라이언트 상호 작용에 대한 응답 시간의 추가 오버헤드가 결정됩니다.

네트워크 구성 계획

Application Server를 네트워크로 통합하는 방법을 계획할 때 대역폭 요구 사항을 예측하고 사용자의 성능 요구 사항에 맞출 수 있는 방식으로 네트워크를 계획합니다.

이 절은 다음 내용으로 구성됩니다.

- 44 페이지 “대역폭 요구 사항 예측”
- 44 페이지 “필요 대역폭 계산”
- 45 페이지 “최대 로드 예측”
- 45 페이지 “서브넷 구성”
- 45 페이지 “네트워크 카드 선택”

- 46 페이지 “HADB를 위한 네트워크 설정”
- 48 페이지 “오류 클래스 확인”

대역폭 요구 사항 예측

원하는 네트워크 크기 및 대역폭을 결정하려면 먼저 네트워크 트래픽을 결정하고 그 최대치를 확인합니다. 전체 양이 최대치에 도달하는 특정 시간, 요일 또는 일자가 있는지 확인하여 트래픽 최대 기간을 결정합니다.

최대 로드 시간 동안 네트워크의 패킷 수는 최고 수준에 도달합니다. 일반적으로 최대 로드를 사용하도록 설계할 경우 최대 양을 100% 처리하도록 시스템 크기를 조정합니다. 그러나 네트워크는 예상치 않게 동작하며, 네트워크 크기를 조정하더라도 최대 양의 100%를 처리하지 못하는 경우가 발생할 수 있다는 사실에 유의해야 합니다.

예를 들어 최대 로드 시 Application Server에 배포된 응용 프로그램에 액세스할 때 사용자의 5%가 간혹 네트워크에 즉시 액세스할 수 없다고 가정합니다. 이 5%의 사용자 중에서 첫 시도 이후 액세스를 다시 시도하는 사용자 수를 예측합니다. 다시 이 사용자 중 일부가 액세스에 실패할 수 있고 이러한 실패한 부분 중 일부 사용자가 또 다시 시도합니다. 따라서 사용자가 계속 액세스를 시도하는 시간 전체에 걸쳐 최대 사용이 분산되기 때문에 최대 로드가 더 오래 나타납니다.

필요 대역폭 계산

37 페이지 “성능 목표 설정”에서 계산한 결과에 따라 사이트에서 Application Server를 배포하는 데 필요한 추가 대역폭을 결정합니다.

액세스 방식(T-1 회선, ADSL, 케이블 모뎀 등)에 따라 예측된 로드를 처리하는 데 필요한 추가 증가 대역폭 크기를 계산합니다. 예를 들어 사이트에서 T-1 또는 더 높은 속도의 T-3 회선을 사용한다고 가정합니다. 대역폭이 제공되면 해당 사이트에서 초당 생성된 평균 요청 수와 최대 로드 최대값에 따라 네트워크에 필요한 회선 수를 예측합니다. 웹 사이트 분석 및 모니터링 도구를 사용하여 이러한 수치를 계산합니다.

예 2–3 필요 대역폭 계산

단일 T-1 회선은 1.544Mbps를 처리할 수 있습니다. 따라서 T-1 회선 4개로 구성된 네트워크는 약 6Mbps의 데이터를 처리할 수 있습니다. 클라이언트로 반환된 평균 HTML 페이지가 30KB인 경우 T-1 회선 4개로 구성된 이 네트워크는 다음과 같은 초당 트래픽을 처리할 수 있습니다.

6,176,000비트/8비트 = 초당 772,000바이트

초당 772,000바이트/30KB = 초당 약 25개의 동시 응답 페이지

초당 25페이지의 트래픽이 있는 이 시스템에서는 하루 내내 균일한 로드 처리량이 제공된다는 가정 하에 시간당 90,000페이지(25 x 60초 x 60분), 따라서 하루에 최대 2,160,000페이지를 처리할 수 있습니다. 최대 로드 최대값이 이 값보다 큰 경우 대역폭을

예 2–3 필요 대역폭 계산 *(계속)*

적절하게 늘립니다.

최대 로드 예측

하루 내내 균일한 로드 처리량을 제공하는 것은 실제로 불가능할 수 있습니다. 최대 로드 발생 시기, 지속 기간 및 총 로드에 대한 최대 로드 비율을 결정해야 합니다.

예 2–4 최대 로드 계산

최대 로드가 2시간 동안 지속되고 전체 로드의 30%인 2,160,000페이지를 차지할 경우 해당 시간 동안 T-1 회선을 통해 648,000페이지를 전송해야 합니다.

따라서 이 2시간 동안 최대 로드를 수용하려면 다음 계산에 따라 T-1 회선 수를 늘려야 합니다.

648,000페이지/120분 = 분당 5,400페이지

분당 5,400페이지/60초 = 초당 90페이지

4개 회선이 초당 25페이지를 처리할 수 있는 경우 해당 페이지 수의 약 4배에는 회선 수의 4배가 필요하므로 이 경우 16개 회선이 필요합니다. 16개의 회선은 30%의 최대 로드 최대값을 실제로 처리할 수 있음을 의미합니다. 나머지 70%의 로드는 하루 중 나머지 시간 동안 이러한 여러 회선을 통해 처리할 수 있습니다.

서브넷 구성

응용 프로그램 서버 인스턴스와 HADB 노드가 각기 다른 호스트 시스템에 있는 개별 계층 토폴로지를 사용할 경우 모든 HADB 노드를 별도의 서브넷에 배치하여 성능을 향상시킬 수 있습니다. 이는 HADB에서 UDP(User Datagram Protocol)를 사용하기 때문입니다. 별도 서브넷을 사용하면 해당 서브넷 외부 시스템에 대한 UDP 트래픽이 감소합니다.그러나 모든 HADB 노드는 동일한 서브넷에 있어야 합니다.

모든 노드 및 관리 에이전트가 같은 서브넷에 있는 한 관리 클라이언트를 다른 서브넷에서 실행할 수 있습니다. 모든 노드 에이전트에서 모든 호스트와 포트에 액세스할 수 있어야 하고 노드가 방화벽, UDP 차단 등으로 차단되면 안 됩니다.

HADB에는 UDP 멀티캐스트가 사용되므로 HADB 노드가 포함된 서브넷은 멀티캐스트를 사용하도록 구성해야 합니다.

네트워크 카드 선택

큰 대역폭과 최적의 네트워크 성능을 위해 Application Server를 호스팅하는 서버와 HADB 노드 사이에 최소 100Mbps의 이더넷 카드 또는 1Gbps 이더넷 카드(권장)를 사용합니다.

HADB를 위한 네트워크 설정

주 – HADB는 UDP 멀티캐스트를 사용하므로 시스템의 라우터와 호스트 네트워크 인터페이스 카드에서 멀티캐스트를 활성화해야 합니다. HADB가 여러 하위 네트워크에 걸쳐 있으면 하위 네트워크 사이의 라우터에서도 멀티캐스트를 활성화해야 합니다. 최상의 결과를 얻으려면 HADB 노드를 모두 같은 네트워크에 배치합니다. Application Server 인스턴스는 다른 하위 네트워크에 존재할 수 있습니다.

다음은 네트워크에서 HADB가 최적의 상태로 작동하기 위한 권장 사항입니다.

- 각각의 네트워크 인터페이스가 100Mbps 이상의 전용 이더넷 채널을 갖도록 스위칭된 라우터를 사용합니다.
- HADB 노드를 4개 이상 호스팅하는 다중 CPU 시스템에서 HADB를 실행하는 경우에는 1Gbps 이더넷 카드를 사용합니다. 평균 세션 크기가 50KB보다 큰 경우에는 시스템당 HADB 개수가 4개 미만인 경우에도 1Gbps 이더넷 카드를 사용합니다.
- HADB 노드 내에서 네트워크 병목 현상이 의심되는 경우 다음을 수행합니다.
  - HADB 서버에서 네트워크 모니터링 소프트웨어를 실행하여 문제를 진단합니다.
  - 네트워크에서 100Mbps 이더넷 카드를 1Gbps 이더넷 카드로 교체합니다.

가용성 계획

이 절은 다음 내용으로 구성되어 있습니다.

가용성 조정

시스템 및 응용 프로그램의 가용성을 계획하려면 여러 응용 프로그램에 액세스하는 사용자 그룹의 가용성 요구 사항을 평가합니다. 예를 들어 외부 유료 사용자와 비즈니스 파트너는 종종 내부 사용자보다 더 높은 QoS(서비스 품질) 기대치를 갖습니다. 따라서 응용 프로그램 기능이나 응용 프로그램, 또는 서버를 사용할 수 없게 되는 경우 내부 사용자가 외부 유료 고객보다 상황을 잘 받아들입니다.

다음 그림은 점점 감소하는 발생 가능한 이벤트를 완화하기 위해 증가하는 비용과 복잡성을 보여줍니다. 연속선상의 한쪽 끝인 로드 균형이 조정된 단순 클러스터의 경우 현지화된 응용 프로그램, 미들웨어 및 하드웨어 오류를 허용합니다. 반대편 끝의 지리적으로 떨어져 있는 클러스터의 경우 전체 데이터 센터에 영향을 주는 주요 재난을 완화할 수 있습니다.

적절한 투자 수익률(ROI) 실현을 위해 응용 프로그램 내 기능의 가용성 요구 사항을 확인하는 작업은 종종 의미가 있습니다. 예를 들어 보험 견적 시스템을 사용할 수 없게 되어 잠재적으로 새 비즈니스를 놓치는 상황은 허용될 수 없지만 기존 고객이 현재 보상 범위를 볼 수 있는 계정 관리 기능의 경우 잠시 사용할 수 없더라도 기존 고객을 잃게 되지는 않습니다.

클러스터를 사용하여 가용성 향상

가장 기본적인 수준의 클러스터는 대개 여러 개의 물리적 서버에서 호스팅되지만 클라이언트에 단일 인스턴스로 나타나는 Application Server 인스턴스의 그룹입니다. 클러스터는 단일 시스템의 단일 인스턴스보다 높은 가용성과 수평 확장성을 제공합니다. 이 기본 수준 클러스터링은 HTTP 및 HTTPS 요청을 수신하여 클러스터 내 Application Server 인스턴스 중 하나로 전달하는 Application Server의 HTTP 로드 밸런서 플러그인과 함께 작동합니다. ORB와 통합 JMS 브로커도 Application Server 클러스터에 대한 로드 균형 조정을 수행합니다. 한 인스턴스에 오류가 발생하거나 네트워크 오류 때문에 사용할 수 없게 되거나 응답하지 않으면 요청이 기존에 있던 사용 가능한 시스템으로만 리디렉션됩니다. 또한 로드 밸런서는 실패한 인스턴스가 복구되었을 때를 인식하고 그에 따라 로드를 재분산할 수 있습니다.

HTTP 로드 밸런서는 서버와 특정 URL을 모니터하여 사용 가능 여부를 확인할 수 있는 **상태 검사기** 프로그램도 제공합니다. 상태 검사 자체가 큰 부담이 되지 않도록 상태 검사 오버헤드 관리에 유의해야 합니다.

상태 비보존형 응용 프로그램이나 중요하지 않은 단순 사용자 트랜잭션만 관련된 응용 프로그램의 경우 로드 균형이 조정된 단순 클러스터만으로 대개 충분합니다. 업무상 중요한 상태 보존형 응용 프로그램의 경우 세션 지속성을 위해 HADB를 사용하는 것이 좋습니다. HADB에 대한 개요를 보려면 Application Server 관리 설명서의 1 장에서 29 페이지 “고가용성 데이터베이스”를 참조하십시오.

응용 프로그램을 온라인으로 업그레이드하려면 Application Server 인스턴스를 여러 클러스터로 그룹화하는 것이 좋습니다. Application Server에는 응용 프로그램과 인스턴스를 모두 **정지**하는 기능이 있습니다. 정지는 현재 인스턴스나 응용 프로그램을 사용하는 사용자에게 영향을 주지 않고 제어된 방식으로 인스턴스(또는 인스턴스 그룹)나 특정 응용 프로그램을 오프라인으로 전환하는 기능입니다. 한 인스턴스가 정지됨에 따라 새 사용자는 다른 인스턴스의 업그레이드된 응용 프로그램을 사용하게 됩니다. 이러한 응용 프로그램 업그레이드 유형을 **롤링 업그레이드**라고 합니다. 활성 응용 프로그램 업그레이드에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 “가용성 손실 없이 응용 프로그램 업그레이드”를 참조하십시오.

시스템에 중복 장치 추가

고가용성을 얻기 위한 한 가지 방법으로 하드웨어 및 소프트웨어를 중복하여 시스템에 추가할 수 있습니다. 한 장치에 오류가 발생하면 중복 장치가 역할을 인계 받게 되는데, 이를 내결함성이라고도 합니다. 일반적으로 고가용성을 극대화하기 위해 시스템의 모든 가능한 오류 지점을 확인 및 제거합니다.

오류 클래스 확인

중복 수준은 시스템이 허용해야 하는 오류 클래스(오류 유형)에 의해 결정됩니다. 다음은 오류 클래스의 몇 가지 예입니다.

- 시스템 프로세스
- 시스템
- 전원 공급 장치
- 디스크
- 네트워크 오류
- 건물 화재 또는 기타 예방 가능한 재해
- 예기치 않은 자연 재난

중복 시스템 프로세스는 단일 시스템 프로세스 오류 및 단일 시스템 오류를 허용합니다. 미러된(쌍을 이룬) 중복 시스템을 다른 전원 공급 장치에 연결할 경우 단일 전원 오류가 허용됩니다. 미러된 시스템을 다른 건물에서 유지할 경우 한 건물에서 발생한 화재를 허용할 수 있습니다. 미러된 시스템을 지리적으로 격리된 위치에서 유지하면 지진과 같은 자연 재난을 허용할 수 있습니다.

HADB 중복 장치를 사용하여 가용성 향상

가용성을 향상시키기 위해 HADB 노드는 37 페이지 "성능 목표 설정"의 설명대로 항상 DRU(Data Redundancy Unit)에서 사용됩니다.

HADB 예비 노드를 사용하여 내결함성 향상

예비 노드를 사용하면 내결함성이 향상됩니다. 예비 노드는 필수 항목은 아니지만 최대 가용성을 제공합니다.

페일오버 용량 계획

페일오버 용량 계획은 서버나 프로세스 오류 시 시스템 중단 없이 데이터를 복구하고 처리를 계속할 수 있도록 Application Server 배포에 추가해야 하는 추가 서버 및 프로세스의 수를 결정하는 작업입니다. 시스템이 오버로드되면 프로세스나 서버 오류가 발생하여 응답 시간 저하나 전체 서비스 손실을 초래할 수 있습니다. 성공적인 배포에는 반드시 이러한 상황에 대한 대비가 있어야 합니다.

특히 최대 로드 시 용량을 유지하려면 Application Server 인스턴스를 실행하는 예비 시스템을 기존 배포에 추가합니다.

예를 들어 각각 한 개의 Application Server 인스턴스를 실행하는 시스템 두 대로 구성된 시스템을 가정할 경우 두 대의 시스템에서 초당 요청이 300개에 달하는 최대 로드를 함께 처리합니다. 두 시스템 간 로드 분산이 균일하다고 했을 때 둘 중 한 대의 시스템을 사용할 수 없게 될 경우 이 시스템은 요청을 150개만 처리할 수 있으므로 최대 로드 시 요청의 절반이 처리되지 않습니다.

설계 결정 사항

설계 결정 사항에는 시스템 설계를 최대 로드에 맞출 것인지 고정 상태 로드에 맞출 것인지에 대한 선택과 다양한 역할의 시스템 수 및 해당 시스템 크기가 포함됩니다.

최대 로드 설계 또는 고정 상태 로드 설계

일반적인 배포에서 고정 상태 작업 로드와 최대 작업 로드 사이에는 다음과 같은 차이가 있습니다.

- 시스템이 최대 로드를 처리하도록 설계된 경우 응답 시간을 저하하지 않고 예상되는 최대 사용자 및 요청 로드를 소화할 수 있습니다. 즉, 시스템에서 극단적인 경우의 예상 시스템 로드를 처리할 수 있습니다. 최대 로드와 고정 상태 로드의 차이가 큰 경우 최대 로드 설계는 자주 유휴 상태가 되는 자원에 대한 비용의 소비를 의미합니다.
- 시스템이 고정 상태 로드를 처리하도록 설계된 경우 예상 최대 로드를 처리하기 위해 필요한 모든 자원을 갖추고 있지 않으므로 최대 로드가 발생할 때 시스템의 응답 시간이 느려집니다.

시스템에서 최대 로드를 처리하게 될 빈도에 따라 설계 기준이 최대 로드 또는 고정 상태 로드로 결정됩니다.

최대 로드가 하루 중에도 여러 번 발생하는 경우에는 최대 로드를 처리할 용량을 확장하는 것이 좋은 선택이 될 수 있습니다. 시스템이 작동 시간의 90%는 고정 상태 로드로 작동하고 나머지 10%의 시간 동안만 최대 로드 상태에서 작동하는 경우에는 고정 상태 로드를 중심으로 설계된 시스템을 배포하는 것이 바람직합니다. 이는 시스템의 응답 시간이 10%에 해당하는 시간 동안만 느려진다는 것을 의미합니다. 시스템이 최대 로드에서 작동하는 빈도나 지속 시간을 확인하여 자원을 시스템에 추가해야 하는지 여부를 결정합니다.

시스템 크기 조정

Application Server 인스턴스의 로드, HADB의 로드 및 페일오버 요구 사항에 따라 다음을 결정할 수 있습니다.

- 50 페이지 “Application Server 인스턴스 수”

- 50 페이지 “HADB 노드 수”
- 51 페이지 “HADB 호스트 수”
- 51 페이지 “HADB 저장 용량”

Application Server 인스턴스 수

각 인스턴스는 둘 이상의 CPU(Central Processing Unit)를 사용할 수 있지만 필요한 Applications Server 인스턴스(호스트) 수를 결정하려면 각 Application Server 인스턴스에 대해 38 페이지 “Application Server 인스턴스의 로드 예측”에 설명된 요소를 기초로 환경을 평가합니다.

HADB 노드 수

일반적인 지침으로 시스템에서 CPU별로 하나의 HADB 노드를 포함하도록 계획합니다. 예를 들어 CPU가 두 개인 하나의 시스템에 HADB 노드 두 개를 사용합니다.

주 – 더 큰 시스템을 사용하는 경우처럼 시스템당 둘 이상의 HADB 노드가 있는 경우에는 시스템에 여러 개의 무정전 전원 공급 장치, 독립 디스크 컨트롤러 등 충분한 중복성과 확장성이 있어야 합니다.

또는 다음 절차를 사용합니다.

▼ 필요한 HADB 노드 수를 결정하는 방법

1 다음 매개 변수를 결정합니다.

- 최대 동시 사용자 수 n_{users}
- 평균 BLOB 크기 s
- 사용자당 최대 처리 속도(NTPS)

2 최대 기본 데이터 양의 크기(GB)를 나타내는 V_{data}를 결정합니다.

다음 공식을 사용합니다.

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

3 최대 HADB 데이터 전송 속도 R_{dt}를 결정합니다.

이 값은 응용 프로그램측에서 HADB로 전송되는 데이터 양을 반영합니다. 다음 공식을 사용합니다.

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

4 노드 수 N_{NODES}를 결정합니다.

다음 공식을 사용합니다.

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

노드는 쌍으로 작동하므로 이 값을 짝수로 반올림합니다.

HADB 호스트 수

데이터 전송 요구 사항에 따라 HADB 호스트 수를 결정합니다. 이 계산에서는 모든 호스트의 하드웨어 구성과 운영 체제가 비슷하고 호스트에서 실행하는 노드를 수용하는 데 필요한 자원이 있다고 가정합니다.

▼ 호스트 수를 계산하는 방법

1 최대 호스트 데이터 전송 속도 R_{max}를 결정합니다.

이 값은 네트워크와 호스트 하드웨어에 따라 결정되므로 경험을 기초로 결정합니다. 이 값은 이전 절에서 결정한 최대 HADB 데이터 전송률 R_{dt}와 다른 값입니다.

2 이 데이터를 수용하는 데 필요한 호스트 수를 결정합니다.

호스트 수 N_{HOSTS}에 분산된 데이터의 양 V를 업데이트하면 각 호스트가 약 $4V/N_{HOSTS}$의 데이터를 수신합니다. 다음 공식으로 이 데이터 양을 수용하는 데 필요한 호스트 수를 결정합니다.

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

각 DRU의 호스트 수를 같게 하기 위해 이 값을 가장 가까운 짝수로 반올림합니다.

3 각 DRU에 예비 노드용 호스트를 하나씩 추가합니다.

다른 호스트가 각기 N개의 데이터 노드를 실행할 경우 이 호스트에서 N개의 예비 노드를 실행하게 합니다. 그러면 N개 데이터 노드를 다운시키는 단일 시스템 오류가 허용됩니다.

각 호스트는 하나 이상의 노드를 실행해야 하므로 노드 수가 호스트 수보다 작을 경우($N_{NODES} < N_{HOSTS}$) N_{NODES}를 N_{HOSTS}와 같아지도록 조정합니다. 노드 수가 호스트 수보다 큰 경우에는($N_{NODES} \> N_{HOSTS}$) 동일한 호스트에서 여러 개의 노드를 실행할 수 있습니다.

HADB 저장 용량

HADB는 네트워크 용량이 초과될 때까지 노드를 추가할 수 있으므로 직선에 가까운 확장이 가능합니다. 각 노드는 전용 디스크의 저장 장치에 구성되어야 합니다. 모든 노드에는 저장 장치에 할당된 크기에 상당하는 공간이 있어야 합니다. 저장 장치가 로컬 디스크에 할당되어 있는지 확인합니다.

예상 세션 데이터 크기를 *x*MB라고 가정합니다. HADB에서는 데이터를 미러 노드에 복제하므로 2*x*MB의 저장소가 필요합니다. 또한 HADB는 색인을 사용하여 데이터에 빠르게 액세스할 수 있도록 합니다. 두 개의 노드에는 색인에 사용할 추가 2*x*MB가 필요하므로 전체 4*x*의 저장 용량이 필요합니다. 따라서 HADB의 예상 저장 용량 요구 사항은 예상 데이터 양의 4배입니다.

새 노드를 추가한 후 데이터를 다시 단편화해야 할 수 있으므로 향후 HADB의 데이터 손실 없는 확장을 고려하면 온라인 업그레이드를 위한 추가 저장 용량을 제공해야 합니다. 이 경우 데이터 장치에 비슷한 크기의 추가 공간(4*x*)이 필요합니다. 따라서 예상 저장 용량은 예상 데이터 양의 8배입니다.

또한 HADB에서는 다음과 같은 디스크 공간이 사용됩니다.

- 로그 버퍼 임시 저장을 위한 공간. 이 공간은 로그 버퍼 크기의 4배입니다. 로그 버퍼는 데이터 관련 작업을 추적합니다. 로그 버퍼 크기 기본값은 48MB입니다.
- 내부 관리용 공간. 이 공간은 저장 장치 크기의 1%입니다.

다음은 *x*MB의 세션 데이터에 대한 HADB 저장 공간 요구 사항을 간략하게 설명한 표입니다.

표 2–3 XMB의 세션 크기에 대한 HADB 저장 공간 요구 사항

조건	필요한 HADB 저장 공간
온라인일 필요가 **없는** 상태에서 HADB 노드 추가 또는 제거	4*x*MB + (4*로그 버퍼 크기) + 장치 크기의 1%
온라인이여야 하는 상태에서 HADB 노드 추가 또는 제거	8*x*MB + (4*로그 버퍼 크기) + 장치 크기의 1%

HADB는 장치 공간이 부족할 경우 데이터를 삽입하거나 업데이트하는 클라이언트 요청을 허용하지 않지만 삭제 작업은 허용합니다. HADB에 장치 공간이 부족하면 오류 코드 4593 또는 4592가 반환되고 해당 오류 메시지가 내역 파일에 기록됩니다. 이러한 메시지에 대한 자세한 내용은 **Sun Java System Application Server 9.1 Error Message Reference**의 14 장, “HADB Error Messages”를 참조하십시오.

Message Queue 브로커 배포 계획

Java Message Service(JMS) API는 J2EE 응용 프로그램 및 구성 요소가 메시지를 작성하고, 보내고, 받고, 읽을 수 있도록 하는 메시징 표준입니다. 또한 느슨하게 결합되고 안정적인 비동기식 분산 통신을 가능하게 합니다. JMS를 구현하는 Sun Java System Message Queue는 Application Server와 통합되어 MDB(Message-Driven Bean)와 같은 구성 요소를 만들 수 있도록 합니다.

Sun Java System Message Queue(MQ)는 J2EE Connector Architecture(JCA) 사양 1.5에서 정의된 자원 어댑터라고도 하는 **커넥터 모듈**을 사용하여 Application Server와 통합됩니다. 커넥터 모듈은 Application Server에 기능을 추가하는 표준화된 방법입니다. Application Server에 배포된 J2EE 구성 요소는 커넥터 모듈을 통해 통합된 JMS 공급자를 사용하여 JMS 메시지를 교환합니다. 기본적으로 JMS 공급자는 Sun Java System Message Queue이지만 원하는 경우 JCA 1.5를 구현하는 다른 JMS 공급자를 사용할 수 있습니다.

Application Server에서 JMS 자원을 만들면 백그라운드에서 커넥터 자원이 만들어집니다. 따라서 각 JMS 작업은 커넥터 런타임을 호출하며 백그라운드에서 MQ 자원 어댑터를 사용합니다.

자원 어댑터 API 사용 외에도 Application Server에서는 추가 MQ API를 사용하여 MQ와 향상된 통합을 제공합니다. 이러한 긴밀한 통합으로 인해 MDB에 대한 커넥터 페일오버, 아웃바운드 연결의 로드 균형 조정 및 인바운드 메시지의 로드 균형 조정과 같은 기능을 사용할 수 있습니다. 이러한 기능은 메시징 트래픽에 내결함성과 고가용성을 제공합니다.

다중 브로커 클러스터

MQ Enterprise Edition에서는 **브로커 클러스터**라고도 하는 여러 개의 상호 연결된 브로커 인스턴스를 사용할 수 있습니다. 브로커 클러스터를 사용하면 클라이언트 연결이 클러스터에 있는 모든 브로커 간에 분산됩니다. 클러스터링은 수평적 확장을 제공하며 가용성을 향상시킵니다.

단일 메시지 브로커는 약 8개까지 CPU를 확장할 수 있으며 일반적인 응용 프로그램에 충분한 처리량을 제공합니다. 브로커 프로세스에 오류가 발생하면 브로커가 자동으로 다시 시작됩니다. 그러나 브로커에 연결된 클라이언트 수가 증가하고 전달되는 메시지 수가 증가함에 따라 브로커는 결국 파일 설명자 수 및 메모리 등의 제한을 초과하게 됩니다.

한 클러스터에 단일 브로커가 아니라 여러 개의 브로커를 사용하면 다음이 가능합니다.

- 단일 시스템에서 하드웨어 오류가 발생해도 메시징 서비스를 제공합니다.
- 시스템 유지 보수를 수행하면서 중단 시간을 최소화합니다.
- 여러 사용자 저장소가 있는 워크그룹을 수용합니다.
- 방화벽 제한을 처리합니다.

그러나 브로커가 여러 개 있어도 브로커 오류 발생 시 진행 중인 트랜잭션이 대체 브로커에서 계속되지 않을 수도 있습니다. MQ에서 실패한 연결을 클러스터 내 다른 브로커와 다시 설정하지만 진행 중인 트랜잭션 메시징과 롤백 트랜잭션이 손실됩니다. 완료하지 못한 트랜잭션을 제외하고 사용자 응용 프로그램에는 영향이 없습니다. 연결을 계속 사용할 수 있으므로 서비스 페일오버가 보장됩니다.

따라서 MQ는 클러스터에서 고가용성 지속성 메시징을 지원하지 않습니다. 브로커가 오류 발생 이후 다시 시작되면 지속성 메시지 전달을 자동으로 복구하여 완료합니다. 지속성 메시지는 파일 시스템 또는 데이터베이스에 저장될 수 있습니다. 그러나 브로커를 호스팅하는 시스템의 하드웨어 고장이 복구되지 않으면 메시지가 손실될 수 있습니다.

Sun Message Queue용 Sun Cluster 데이터 서비스가 설치된 Solaris 플랫폼에서는 지속성 메시지의 투명한 페일오버를 지원합니다. 이 구성은 Sun Cluster의 전역 파일 시스템과 IP 페일오버를 사용하여 신뢰할 수 있는 고가용성을 제공하며 Java Enterprise System에 포함되어 있습니다.

마스터 브로커와 클라이언트 동기화

다중 브로커 구성에서 각 대상은 클러스터 내 모든 브로커에서 복제됩니다. 각 브로커는 다른 모든 브로커의 대상에 등록된 메시지 사용자를 인식합니다. 따라서 각 브로커는 직접 연결된 해당 고유 메시지 생성자에서 원격 메시지 사용자로 메시지를 라우팅하고 원격 생성자에서 직접 연결된 해당 고유 사용자로 메시지를 전달합니다.

클러스터 구성에서 각 메시지가 생성자가 직접 연결된 브로커는 해당 생성자에 의해 브로커에 전송된 메시지의 라우팅을 수행합니다. 따라서, 지속성 메시지는 메시지의 **홈 브로커**에 의해 저장 및 라우팅됩니다.

관리자가 브로커에서 대상을 만들거나 삭제할 때마다 이 정보는 클러스터에 속한 다른 모든 브로커에 자동으로 전파됩니다. 마찬가지로 메시지 사용자가 홈 브로커에 등록될 때마다 또는 명시적으로 지정되었거나 클라이언트나 네트워크 오류, 홈 브로커 다운으로 인해 메시지 사용자가 홈 브로커에서 연결이 끊어질 때마다 해당 메시지 사용자에 대한 관련 정보가 클러스터 전체에 전파됩니다. 영구 가입 정보도 비슷한 방식으로 클러스터의 모든 브로커에 전파됩니다.

Message Queue 브로커를 사용하도록 Application Server 구성

Application Server의 Java Message Service는 Message Queue에 대한 커넥터 모듈(자원 어댑터)을 나타냅니다. 관리 콘솔이나 `asadmin` 명령줄 유틸리티를 사용하여 Java Message Service를 관리할 수 있습니다.

MQ 브로커(JMS 호스트)는 Application Server 프로세스와 다른 JVM에서 실행됩니다. 따라서 여러 Application Server 인스턴스나 클러스터가 같은 MQ 브로커 집합을 공유할 수 있습니다.

Application Server에서 *JMS* **호스트**는 MQ 브로커를 나타냅니다. Application Server의 Java Message Service 구성에는 사용할 모든 JMS 호스트가 포함된 JMS 호스트 목록(AddressList)이 있습니다.

관리 콘솔을 사용하여 JMS 관리

관리 콘솔에서 Java Message Service 노드를 사용하여 특정 구성에 대한 JMS 등록 정보를 설정할 수 있습니다. 다시 연결 간격 및 다시 연결 시도와 같은 등록 정보를 설정할 수 있습니다. 자세한 내용은 **Sun Java System Application Server 9.1 관리 설명서**의 4 장, “JMS(Java Message Service) 자원 구성”을 참조하십시오.

Java Message Service 노드 아래의 JMS 호스트 노드에는 JMS 호스트 목록이 포함되어 있습니다. 목록에서 호스트를 추가 및 제거할 수 있습니다. 각 호스트에 대한 호스트 이름, 포트 번호, 그리고 관리 사용자 이름 및 비밀번호를 설정할 수 있습니다. 기본적으로 JMS 호스트 목록에는 Application Server와 통합된 로컬 MQ 브로커를 나타내는 “default_JMS_host” 라는 MQ 브로커가 포함되어 있습니다.

클러스터의 모든 MQ 브로커를 포함하도록 JMS 호스트 목록을 구성합니다. 예를 들어 세 개의 MQ 브로커가 포함된 클러스터를 설정하려면 브로커별로 Java Message Service 내에 JMS 호스트를 추가합니다. Message Queue 클라이언트는 Java Message Service의 구성 정보를 사용하여 MQ 브로커와 통신합니다.

asadmin을 사용하여 JMS 관리

관리 콘솔 외에도 `asadmin` 명령줄 유틸리티를 사용하여 Java Message Service와 JMS 호스트를 관리할 수 있습니다. 다음 `asadmin` 명령을 사용합니다.

- Java Message Service 속성 구성: `asadmin set`
- JMS 호스트 관리:
  - `asadmin create-jms-host`
  - `asadmin delete-jms-host`
  - `asadmin list-jms-hosts`
- JMS 자원 관리:
  - `asadmin create-jms-resource`
  - `asadmin delete-jms-resource`
  - `asadmin list-jms-resources`

    이러한 명령에 대한 자세한 내용은 **Sun Java System Application Server 9.1 Reference Manual**이나 해당 설명서 페이지를 참조하십시오.

Java Message Service 유형

Application Server와 MQ 브로커 간 통합에는 로컬과 원격의 두 가지 유형이 있습니다. 관리 콘솔의 Java Message Service 페이지에서 이 유형 속성을 설정할 수 있습니다.

로컬 Java Message Service

유형 속성이 LOCAL인 경우 Application Server에서 MQ 브로커를 시작 및 중지합니다. Application Server가 시작될 때 기본 JMS 호스트로 지정된 MQ 브로커를 시작합니다. 마찬가지로 Application Server 인스턴스가 종료될 때 MQ 브로커를 종료합니다. 독립 실행형 Application Server 인스턴스에는 LOCAL 유형이 가장 적절합니다.

LOCAL 유형을 사용할 경우 Start Arguments 속성을 사용하여 MQ 브로커 시작 매개변수를 지정합니다.

원격 Java Message Service

유형 속성이 REMOTE인 경우 Application Server에서는 외부에서 구성된 브로커나 브로커 클러스터를 사용합니다. 이 경우 Application Server와는 별도로 MQ 브로커를 시작 및 중지해야 하며 MQ 도구를 사용하여 브로커 또는 브로커 클러스터를 구성하고 조정해야 합니다. REMOTE 유형은 Application Server 클러스터에 가장 적절합니다.

REMOTE 유형을 사용할 경우 MQ 도구를 사용하여 MQ 브로커 시작 매개 변수를 지정해야 합니다. Start Arguments 속성은 무시됩니다.

기본 JMS 호스트

관리 콘솔의 Java Message Service 페이지에서 기본 JMS 호스트를 지정할 수 있습니다. Java Message Service 유형이 LOCAL인 경우 Application Server 인스턴스가 시작될 때 Application Server에서 기본 JMS 호스트를 시작합니다.

MQ 브로커 클러스터를 사용하려면 기본 JMS 호스트를 삭제한 다음 클러스터의 모든 MQ 브로커를 JMS 호스트로 추가합니다. 이 경우 JMS 호스트 목록의 첫 번째 JMS 호스트가 기본 JMS 호스트가 됩니다.

JMS 호스트 중 하나를 기본 JMS 호스트로 지정하여 설정할 수도 있습니다. Application Server에서 Message Queue 클러스터를 사용할 경우 기본 JMS 호스트가 MQ 특정 명령을 실행합니다. 예를 들어 MQ 브로커 클러스터에 대한 물리적 대상을 만들 때 기본 JMS 호스트는 물리적 대상을 만드는 명령을 실행하지만 클러스터의 모든 브로커는 물리적 대상을 사용합니다.

배포 시나리오 예

메시징 요구 사항을 수용하려면 배포, 성능 및 가용성 요구에 맞게 Java Message Service와 JMS 호스트 목록을 수정합니다. 다음 절에서는 몇 가지 일반적인 시나리오에 대해 설명합니다.

최상의 가용성을 얻기 위해서는 메시징 요구 사항이 Application Server와 맞지 않을 경우 MQ 브로커와 Application Server를 서로 다른 시스템에 배포하거나, 메시징 용량이 충분해질 때까지 각 시스템에서 Application Server 인스턴스와 MQ 브로커를 실행합니다.

기본 배포

Application Server를 설치하면 DAS(Domain Administration Server)가 자동으로 생성됩니다. 기본적으로 DAS의 Java Message Service 유형은 LOCAL입니다. 따라서 DAS를 시작하면 기본 MQ 브로커도 시작됩니다.

새 도메인을 만들면 새 브로커도 생성됩니다. 기본적으로 독립 실행형 서버 인스턴스나 클러스터를 도메인에 추가하면 해당 Java Message Service는 REMOTE로 구성되며 DAS에 의해 시작되는 브로커가 기본 JMS 호스트가 됩니다.

56 페이지 “기본 배포”에서는 세 개의 인스턴스가 포함된 Application Server 클러스터를 사용한 기본 배포의 예를 보여줍니다.

Application Server 클러스터와 함께 MQ 브로커 클러스터 사용

MQ 브로커 클러스터를 사용하도록 Application Server를 구성하려면 Application Server의 Java Message Service에서 모든 MQ 브로커를 JMS 호스트로 추가합니다. 생성된 모든 JMS 연결 팩토리와 배포된 MDB는 지정된 JMS 구성을 사용합니다.

다음 그림은 브로커 클러스터의 MQ 브로커 세 개와 클러스터의 Application Server 인스턴스 세 개를 사용하는 배포 예를 보여줍니다.

응용 프로그램 특정 MQ 브로커 클러스터 지정

경우에 따라 응용 프로그램은 Application Server 클러스터에서 사용하는 MQ 브로커 클러스터와 다른 클러스터를 사용해야 할 수 있습니다. 57 페이지 "응용 프로그램 특정 MQ 브로커 클러스터 지정"에서는 이러한 시나리오의 예를 보여줍니다. 이를 위해서는 JMS 연결 팩토리의 `AddressList` 등록 정보나 MDB 배포 설명자의 `activation-config` 요소를 사용하여 MQ 브로커 클러스터를 지정합니다.

연결 팩토리 구성에 대한 자세한 내용은 **Sun Java System Application Server 9.1 관리 설명서**의 "JMS 연결 팩토리"를 참조하십시오. MDB에 대한 자세한 내용은 **Sun Java System Application Server 9.1 Developer's Guide**의 "Using Message-Driven Beans"를 참조하십시오.

응용 프로그램 클라이언트

응용 프로그램 클라이언트나 독립 실행형 응용 프로그램에서 JMS 관리 대상 객체에 처음 액세스하면 클라이언트 JVM이 서버에서 Java Message Service 구성을 검색합니다. 클라이언트 JVM을 다시 시작해야 JMS 서비스의 추가 변경 사항을 사용할 수 있습니다.

◆ ◆ ◆ 3 장

3 토폴로지 선택

1 장의 설명대로 성능 관련 요소를 예측한 후 Application Server. 토폴로지는 시스템, Application Server 인스턴스 및 HADB 노드의 배열이자 이들 간의 통신 흐름입니다.

두 개의 기본적인 배포 토폴로지가 있습니다. 두 토폴로지에는 모두 공통 빌딩 블록인 클러스터에 포함된 여러 Application Server 인스턴스, 미러된 HADB 노드 집합 및 HADB 예비 노드가 있습니다. 두 토폴로지는 모두 공통 구성 설정 세트가 있어야 제대로 작동할 수 있습니다.

이 장은 다음 내용으로 구성됩니다.

- 두 토폴로지에 대한 59 페이지 “공통 요구 사항”
- 두 토폴로지:
  - 61 페이지 “공존 토폴로지” - Application Server 인스턴스와 HADB 노드가 동일한 시스템에 있습니다.
  - 65 페이지 “개별 계층 토폴로지” - Application Server 인스턴스와 HADB 노드가 서로 다른 시스템에 있습니다.
- 69 페이지 “사용할 토폴로지 결정”

공통 요구 사항

이 절에서는 두 토폴로지의 공통적인 요구 사항에 대해 설명합니다.

- 59 페이지 “일반 요구 사항”
- 60 페이지 “HADB 노드 및 시스템”
- 61 페이지 “로드 밸런서 구성”

일반 요구 사항

두 토폴로지 모두 다음 일반 요구 사항을 충족해야 합니다.

- HADB 노드를 호스팅하는 시스템은 쌍으로 존재해야 합니다. 즉, 시스템 수가 짝수여야 합니다.
- 각 DRU(Data Redundancy Unit)에는 같은 수의 시스템이 있어야 합니다. 미러된(쌍을 이룬) 노드를 기본 노드와 다른 DRU에 배치하여 HADB 데이터베이스를 만듭니다.
- HADB를 호스팅하는 각 시스템에는 모든 영구 정보를 HADB에 저장하기 위해 사용되는 로컬 디스크 저장소가 있어야 합니다.
- HADB 노드를 호스팅하는 시스템에서는 동일한 운영 체제를 실행해야 합니다. 구성 및 성능이 동일하거나 거의 비슷한 시스템을 사용하는 것이 좋습니다.
- HTTP 및 SFSB 세션 정보를 HADB에 영구 저장하려면 Application Server 인스턴스가 클러스터에 있어야 하고 모든 관련 요구 사항을 충족해야 합니다. 클러스터 구성에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 6장, "Application Server 클러스터 사용"을 참조하십시오.
- Application Server 인스턴스를 호스팅하는 시스템은 구성 및 성능이 가능한 한 동일해야 합니다. 이는 로드 밸런서 플러그인이 로드 균형 조정에 대한 라운드 로빈 정책을 사용하기 때문이며 여러 클래스의 시스템에서 인스턴스를 호스팅할 경우에는 최적의 방식으로 로드 균형이 조정되지 않습니다.
- 각 DRU에 대해 별도의 UPS(무정전 전원 공급 장치)를 사용하는 것이 좋습니다.

HADB 노드 및 시스템

각 DRU는 HADB의 전체 데이터 복사본을 포함하므로 다른 DRU를 사용할 수 없게 되는 경우 요청을 계속 처리할 수 있습니다. 그러나 한 DRU의 노드와 다른 DRU에 있는 해당 미러가 동시에 실패하면 일부 데이터가 손실됩니다. 이러한 이유에서 정전이나 디스크 오류와 같은 단일 오류로 두 DRU가 모두 영향을 받지 않도록 시스템을 설정하는 것이 중요합니다.

주 – 각 DRU는 완전히 독립적인 중복 시스템에서 실행되어야 합니다.

HADB 노드와 시스템을 설정할 때 다음 지침을 따릅니다.

- 용량과 처리량을 늘리려면 DRU당 한 노드씩 노드를 쌍으로 추가합니다.
- 예비 노드 수가 각 시스템에서 실행되는 노드 수와 같아지도록 각 DRU를 설정합니다. 이는 구성의 각 시스템에서 *n*개의 데이터 노드를 실행하는 경우 단일 시스템 오류가 발생했을 때 *n*개 노드가 다운되기 때문입니다.
- 모든 시스템에서 같은 수의 HADB 노드를 실행하여 가능한 균일하게 로드 균형을 조정합니다.

주의 – 같은 시스템의 서로 다른 DRU에서 노드를 실행하지 마십시오. 같은 시스템의 서로 다른 DRU에서 노드를 실행해야 할 경우에는 시스템에서 단일 오류 지점(디스크, 메모리, CPU, 전원, 운영 체제 충돌 등과 관련된 오류)을 처리할 수 있는지 확인합니다.

로드 밸런서 구성

두 토폴로지 모두 클러스터에 Application Server 인스턴스가 있습니다. 이러한 인스턴스는 세션 정보를 HADB에 영구 저장합니다. 클러스터의 모든 Application Server 인스턴스에 대한 구성 정보를 포함하도록 로드 밸런서를 구성합니다.

클러스터 설정 및 클러스터에 Application Server 인스턴스를 추가하는 방법에 대한 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 6 장, "Application Server 클러스터 사용"을 참조하십시오.

공존 토폴로지

공존 토폴로지에서 Application Server 인스턴스와 HADB 노드는 같은 시스템에 있습니다(이하 **공존**). 이 토폴로지에는 개별 계층 토폴로지보다 필요한 시스템 수가 적습니다. 공존 토폴로지에서는 CPU를 보다 효율적으로 사용합니다. Application Server 인스턴스와 HADB 노드가 한 시스템을 공유하고 시스템 간에 프로세스가 균일하게 분산됩니다.

이 토폴로지에는 최소한 두 개의 시스템이 필요합니다. 처리량을 향상시키려면 시스템을 쌍으로 추가합니다.

주 – 시스템의 처리 능력을 완전히 활용할 수 있으므로 공존 토폴로지는 대규모 SMP(대칭적 다중 처리) 시스템에 적합합니다.

구성 예

다음 그림은 공존 토폴로지 구성의 예를 보여줍니다.

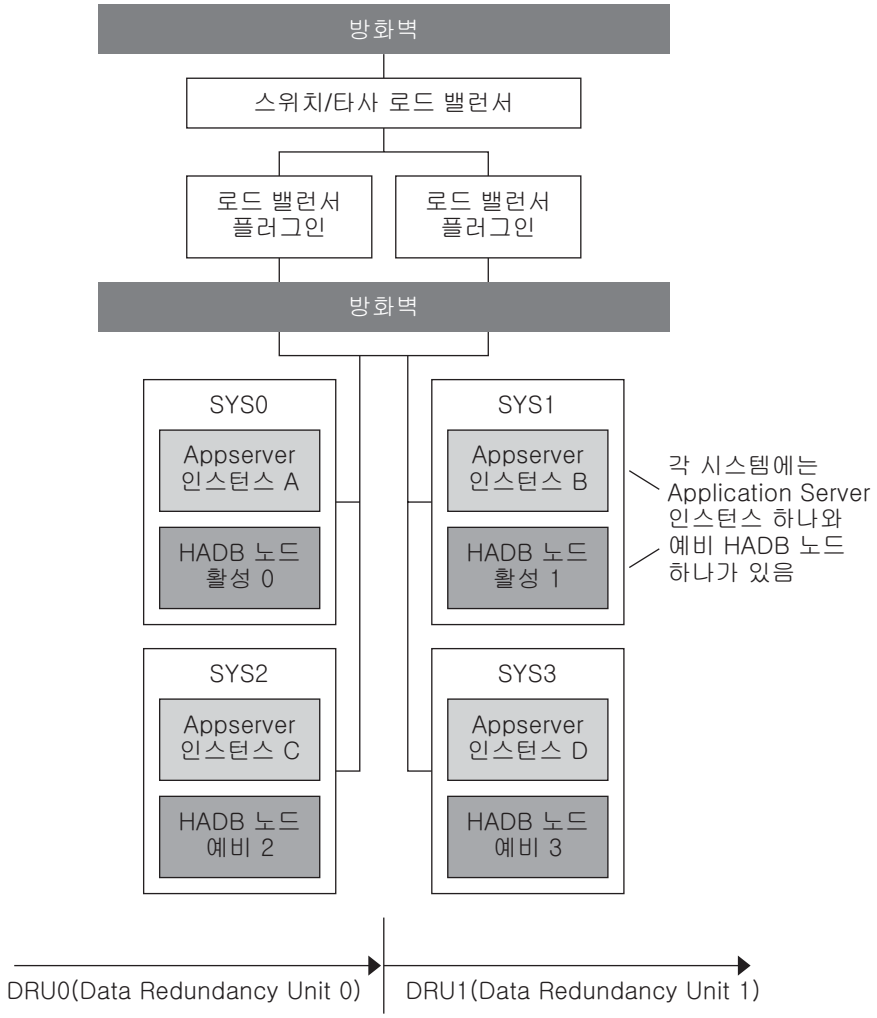

그림 3–1 공존 토폴로지 예

시스템 SYS0은 Application Server 인스턴스 A를, 시스템 SYS1은 Application Server 인스턴스 B를, 시스템 SYS2는 Application Server 인스턴스 C를, 시스템 SYS3은 Application Server 인스턴스 D를 각각 호스팅합니다.

이러한 4개 인스턴스가 정보를 두 개의 DRU에 영구 저장하는 클러스터를 구성합니다.

- **DRU0**은 두 개의 시스템 SYS0 및 SYS2로 구성됩니다. 활성 0 HADB 노드는 시스템 SYS0에 있습니다. 예비 2 HADB 노드는 시스템 SYS2에 있습니다.
- **DRU1**은 두 개의 시스템 SYS1 및 SYS3으로 구성됩니다. 활성 1 HADB 노드는 시스템 SYS1에 있습니다. 예비 3 HADB 노드는 시스템 SYS3에 있습니다.

공존 토폴로지 변형

확장성과 처리량을 향상시키려면 시스템을 추가하여 Application Server 인스턴스와 HADB 노드 수를 증가시킵니다. 예를 들어 각각 하나의 Application Server 인스턴스와 하나의 HADB 노드를 포함하는 두 개의 시스템을 추가할 수 있습니다. 각 DRU에 하나의 노드를 할당하여 HADB 노드를 쌍으로 추가해야 합니다. 63 페이지 "공존 토폴로지 변형"에서는 이 구성에 대해 설명합니다.

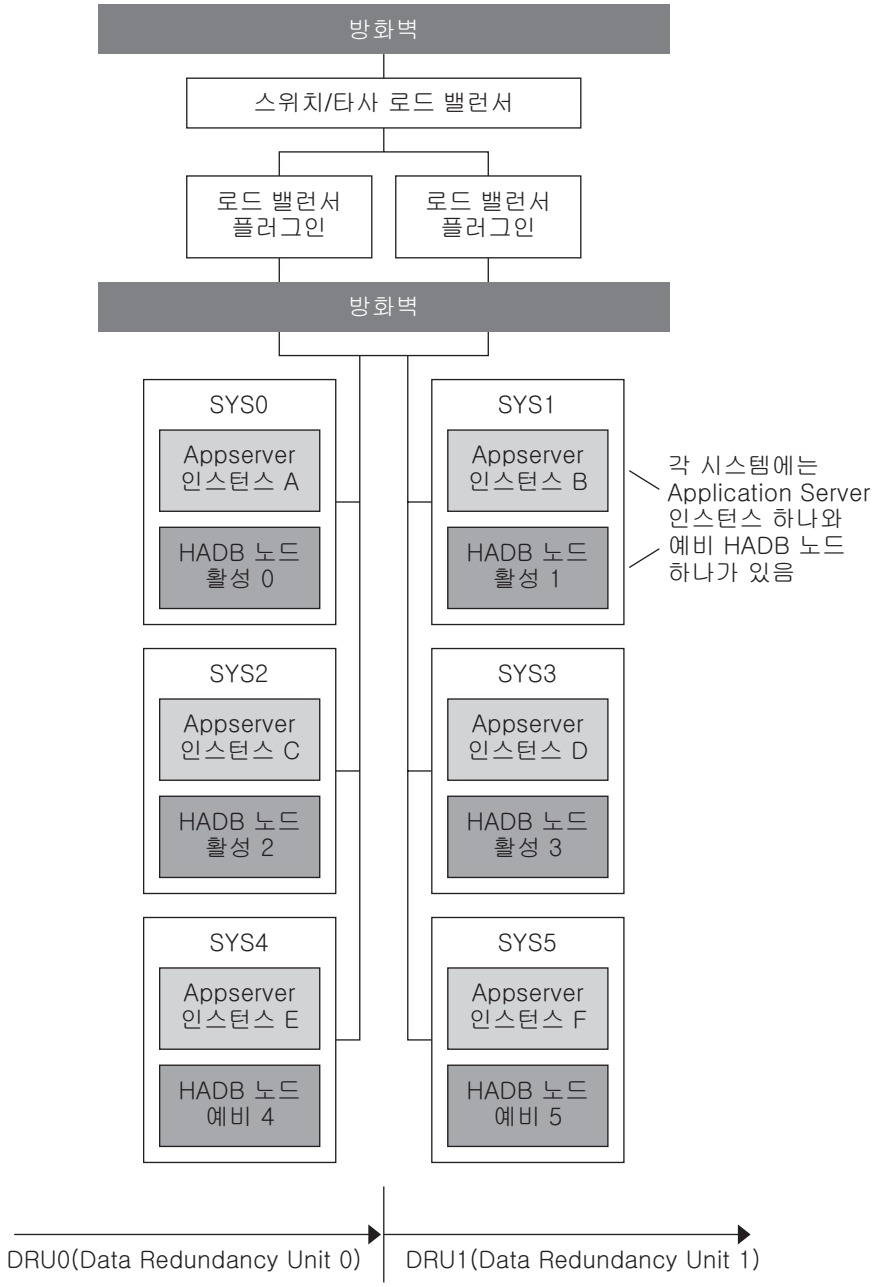

그림 3–2 공존 토폴로지 변형

이 변형에서는 시스템 SYS4 및 SYS5가 61 페이지 "구성 예"에 설명된 공존 토폴로지에 추가되었습니다.

Application Server 인스턴스는 다음과 같이 호스팅됩니다.

- 시스템 SYS0이 인스턴스 A를 호스팅합니다.
- 시스템 SYS1이 인스턴스 B를 호스팅합니다.
- 시스템 SYS2가 인스턴스 C를 호스팅합니다.
- 시스템 SYS3이 인스턴스 D를 호스팅합니다.
- 시스템 SYS4가 인스턴스 E를 호스팅합니다.
- 시스템 SYS5가 인스턴스 F를 호스팅합니다.

이러한 인스턴스가 정보를 두 개의 DRU에 영구 저장하는 클러스터를 구성합니다.

- **DRU0**은 시스템 SYS0, SYS2 및 SYS4로 구성됩니다. 활성 0 HADB 노드는 시스템 SYS0에 있습니다. 활성 2 HADB 노드는 시스템 SYS2에 있습니다. 예비 4 HADB 노드는 시스템 SYS4에 있습니다.
- **DRU1**은 시스템 SYS1, SYS3 및 SYS5로 구성됩니다. 활성 1 HADB 노드는 시스템 SYS1에 있습니다. 활성 3 HADB 노드는 시스템 SYS3에 있습니다. 예비 5 HADB 노드는 시스템 SYS5에 있습니다.

개별 계층 토폴로지

이 토폴로지에서 Application Server 인스턴스와 HADB 노드는 서로 다른 시스템(이하 **개별 계층**)에 있습니다.

이 토폴로지에는 공존 토폴로지보다 큰 하드웨어가 필요합니다. 이 토폴로지는 다양한 유형의 시스템이 있는 경우 적합할 수 있습니다. 한 시스템 집합은 Application Server 인스턴스를 호스팅하도록 할당하고 다른 집합은 HADB 노드를 호스팅하도록 할당할 수 있습니다. 예를 들어 Application Server 인스턴스에는 더 강력한 시스템을 사용하고 HADB에는 상대적으로 성능이 낮은 시스템을 사용할 수 있습니다.

구성 예

다음 그림은 개별 계층 토폴로지를 보여줍니다.

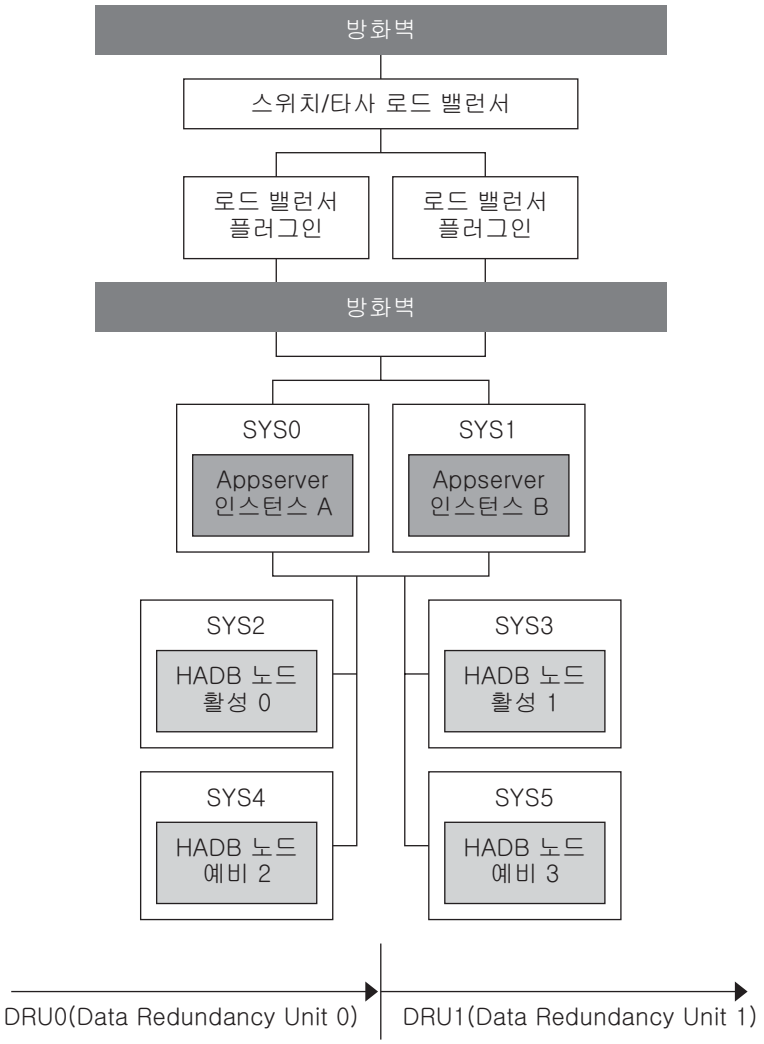

그림 3–3 개별 계층 토폴로지 예

이 토폴로지에서 시스템 SYS0은 Application Server 인스턴스 A를 호스팅하고 시스템 SYS1은 Application Server 인스턴스 B를 호스팅합니다. 이러한 두 인스턴스가 두 개의 DRU에 세션 정보를 영구 저장하는 클러스터를 구성합니다.

- **DRU0**은 두 개의 시스템 SYS2 및 SYS4로 구성됩니다. 활성 0 HADB 노드는 시스템 SYS2에 있고 예비 2 HADB 노드는 시스템 SYS4에 있습니다.
- **DRU1**은 두 개의 시스템 SYS3 및 SYS5로 구성됩니다. 활성 1 HADB 노드는 시스템 SYS3에 있고 예비 3 HADB 노드는 시스템 SYS5에 있습니다.

DRU의 모든 노드는 서로 다른 시스템에 있으므로 한 시스템에 오류가 발생해도 DRU의 전체 데이터를 다른 시스템에서 계속 사용할 수 있습니다.

개별 계층 토폴로지 변형

개별 계층 토폴로지의 변형은 구성에 같은 수준의 시스템을 추가하여 Application Server 인스턴스 수를 증가시킵니다. 예를 들어 새 Application Server 인스턴스를 만들어 다른 시스템을 구성 예에 추가합니다. 마찬가지로 HADB 노드를 호스팅하는 시스템을 추가하여 HADB 노드 수를 증가시킵니다. 각 DRU에 한 노드씩 쌍으로 HADB 노드를 추가해야 합니다.

67 페이지 “개별 계층 토폴로지 변형”에서는 이 구성에 대해 설명합니다.

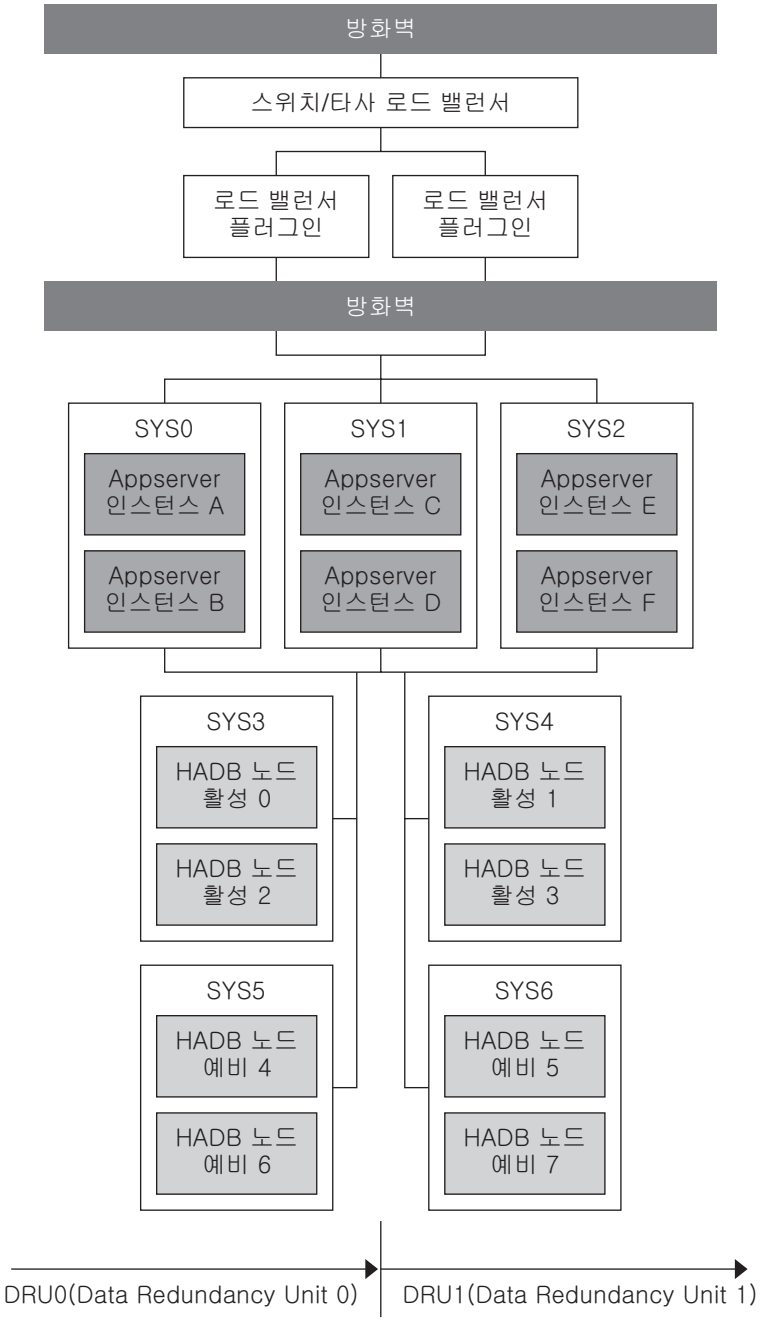

그림 3–4 개별 계층 토폴로지 변형

이 구성에서 Application Server 인스턴스를 호스팅하는 각 시스템에는 두 개의 인스턴스가 있습니다. 따라서 클러스터에 총 6개의 Application Server 인스턴스가 있습니다.

HADB 노드는 시스템 SYS3, SYS4, SYS5 및 SYS6에 있습니다.

DRU0은 두 개의 시스템으로 구성됩니다.

- 활성 0 HADB 노드 및 활성 2 HADB 노드를 호스팅하는 SYS3
- 예비 4 HADB 노드 및 예비 6 HADB 노드가 있는 SYS5

DRU1은 두 개의 시스템으로 구성됩니다.

- 활성 1 HADB 노드 및 활성 3 HADB 노드를 호스팅하는 SYS4
- 예비 5 HADB 노드 및 예비 7 HADB 노드를 호스팅하는 SYS6

HADB 노드를 호스팅하는 각 시스템에서는 두 개의 노드를 호스팅합니다. 따라서 총 8개의 HADB 노드인 활성 노드 4개와 예비 노드 4개가 있습니다.

사용할 토폴로지 결정

성능 및 가용성 요구 사항에 가장 잘 맞는 토폴로지(또는 변형)를 결정하려면 토폴로지를 테스트하고 여러 시스템 및 CPU 조합으로 시험합니다.

목표를 달성하는 데 필요한 절충 조건을 결정합니다. 예를 들어 간편한 유지 보수가 중요한 경우에는 개별 계층 토폴로지가 더 적합합니다. 절충 조건은 토폴로지에 공존 토폴로지보다 많은 시스템이 필요하다는 것입니다.

토폴로지 선택에 있어 중요한 요소는 사용 가능한 시스템 유형입니다. 시스템에 대규모 SMP(대칭적 다중 처리) 시스템이 있는 경우에는 이러한 시스템의 처리 능력을 최대한 활용할 수 있기 때문에 공존 토폴로지가 적합합니다. 시스템에 다양한 시스템 유형이 있는 경우에는 Application Server 계층과 HADB 계층에 서로 다른 시스템 집합을 할당할 수 있기 때문에 개별 계층 토폴로지가 더 유리할 수 있습니다. 예를 들어 Application Server 계층에 가장 강력한 시스템을 사용하고 HADB 계층에 상대적으로 성능이 낮은 시스템을 사용할 수 있습니다.

토폴로지 비교

다음은 공존 토폴로지와 개별 계층 토폴로지를 비교한 표입니다. 왼쪽 열에는 토폴로지 이름, 중간 열에는 토폴로지의 장점, 그리고 오른쪽 열에는 토폴로지의 단점이 각각 나열되어 있습니다.

표 3–1 토폴로지 비교

토폴로지	장점	단점
공존 토폴로지	더 적은 시스템 필요. HADB 노드와 Application Server 인스턴스가 같은 계층에 있으므로 각 예비 노드에 Application Server 인스턴스를 만들어 추가 로드를 처리할 수 있습니다. 향상된 CPU 사용률. 하나의 시스템을 공유하는 Application Server 인스턴스와 HADB 노드 사이에 프로세스가 균일하게 분산됩니다. 처리 능력을 완전히 활용하기 때문에 대규모 SMP(대칭적 다중 처리) 시스템에 유용합니다.	유지 보수 복잡성 증가. 예를 들어 유지 보수를 수행하기 위해 HADB 노드를 호스팅하는 시스템을 종료해야 할 경우 해당 시스템의 Application Server 인스턴스도 사용할 수 없게 됩니다.
개별 계층 토폴로지	간편한 유지 보수. 예를 들어 HADB 노드를 중단할 필요 없이 Application Server 인스턴스를 호스팅하는 시스템에 대한 유지 보수를 수행할 수 있습니다. 시스템 유형이 다양할 경우 유리. Application Server 계층과 HADB 계층에 서로 다른 시스템 집합을 할당할 수 있습니다. 예를 들어 Application Server 계층에 더 강력한 시스템을 사용하고 HADB 계층에 상대적으로 성능이 낮은 시스템을 사용할 수 있습니다.	공존 토폴로지보다 많은 시스템 필요. Application Server 인스턴스와 HADB 노드가 각각 개별 계층에 있기 때문에 Application Server 인스턴스가 HADB 예비 노드를 호스팅하는 시스템에 존재할 수 없습니다. CPU 사용률 감소. Application Server 계층과 HADB 계층의 로드가 균일하지 않을 수 있습니다. 이러한 단점은 시스템 수가 적은 경우(4 - 6개) 영향이 더 큽니다.

◆ ◆ ◆ 4 장

4 배포 확인 목록

이 부록에서는 Application Server에서 평가와 작업을 시작하기 위한 확인 목록을 제공합니다.

배포 확인 목록

표 4–1 확인 목록

구성 요소/기능	설명
응용 프로그램	배포할 응용 프로그램에 대한 다음 요구 사항을 확인합니다. ■ 필요/허용 응답 시간 ■ 최대 로드 특징 ■ 필요한 지속성 범위 및 빈도 ■ `web.xml`의 세션 시간 초과 ■ 페일오버 및 가용성 요구 사항 자세한 내용은 **Sun Java System Application Server 9.1 Performance Tuning Guide**를 참조하십시오.
하드웨어	■ HADB 노드를 호스팅하는 하드웨어와 동일한 유형을 사용합니다. ■ 필요한 크기의 하드 디스크 공간 및 메모리를 설치합니다. ■ 크기 조정 연습을 사용하여 배포 요구 사항을 확인합니다. 자세한 내용은 **Sun Java System Application Server 9.1 릴리스 노트**를 참조하십시오.
운영 체제	■ 제품이 지원되는 플랫폼에 설치되어 있는지 확인합니다. ■ 패치 수준이 최신 상태이고 정확한지 확인합니다. 자세한 내용은 **Sun Java System Application Server 9.1 릴리스 노트**를 참조하십시오.

표 4–1 확인 목록 *(계속)*

구성 요소/기능	설명
네트워크 인프라	■ 단일 실패 지점을 확인하고 해결합니다. ■ NIC 및 기타 네트워크 구성 요소가 제대로 구성되어 있는지 확인합니다. ■ `ttcp` 벤치마크 테스트를 실행하여 처리량이 요구 사항/예상 결과에 부합하는지 확인합니다. ■ HADB 노드가 제대로 설치되도록 `rsh/ssh` 기반 기본 설정을 구성합니다. 자세한 내용은 **Sun Java System Application Server 9.1 Installation Guide**를 참조하십시오.
백엔드 및 기타 외부 데이터 소스	도메인 전문가나 공급업체와 함께 이러한 데이터 소스가 제대로 구성되어 있는지 확인합니다.
시스템 변경 사항/구성	■ 성능/부하 테스트를 실행하기 전에 `/etc/system` 및 해당 Linux 항목의 변경 사항이 완료되었는지 확인합니다. ■ TCP/IP 설정 변경 사항이 완료되었는지 확인합니다. ■ 기본적으로 시스템에는 많은 서비스가 미리 구성되어 제공됩니다. 일부 서비스는 실행할 필요가 없습니다. 필요하지 않은 서비스를 해제하여 시스템 자원을 절약합니다. ■ Solaris의 경우 `Setoolkit`를 사용하여 시스템 동작을 확인합니다. 표시되는 플래그를 확인합니다. 자세한 내용은 **Sun Java System Application Server 9.1 Performance Tuning Guide**를 참조하십시오.
Application Server 및 HADB 설치	■ 이 서버가 NFS 마운트 볼륨에 설치되어 있지 않은지 확인합니다. ■ Application Server와 HADB 노드를 동일한 시스템에 설치할 경우 충분한 디스크 공간과 RAM이 있는지 확인합니다. ■ 여러 HADB 노드를 동일한 시스템에 설치할 경우 충분한 독립 디스크가 있는지 확인합니다. 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 2 장, “고가용성 데이터베이스 설치 및 설정”을 참조하십시오.

표 4–1 확인 목록 (계속)

구성 요소/기능	설명
HADB 구성	■ HADB 데이터 장치의 크기를 설정합니다. ■ DataBufferPoolSize를 정의합니다. ■ LogBufferSize를 정의합니다. ■ InternalBufferSize를 정의합니다. ■ NumberOfLocks를 설정합니다. ■ 다양한 Application Server 구성 요소에 대해 최적의 시간 초과 값을 설정합니다. ■ 파일 시스템에서 HADB 노드의 물리적 레이아웃을 만듭니다. 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 "HADB 구성"을 참조하십시오.
Application Server 구성	■ 로깅: 액세스 로그 회전 사용 가능 ■ 올바른 로깅 수준을 선택합니다. 일반적으로 WARNING(경고)이 적절합니다. ■ 관리 콘솔을 사용하여 J2EE 컨테이너를 구성합니다. ■ 관리 콘솔을 사용하여 HTTP Listener를 구성합니다. ■ 관리 콘솔을 사용하여 ORB 스레드 풀을 구성합니다. ■ 원시 코드와 관련된 Type2 드라이버 또는 호출을 사용할 경우 LD_LIBRARY_PATH에 mtmalloc.so가 지정되어 있는지 확인합니다. ■ 적절한 지속성 범위 및 빈도가 사용되고 개별 웹/EJB 모듈에서 대체되지 않는지 확인합니다. ■ SFSB의 중요 메소드만 검사점이 지정되어 있는지 확인합니다. 조정에 대한 자세한 내용은 **Sun Java System Application Server 9.1 Performance Tuning Guide**를 참조하십시오. 구성에 대한 자세한 내용은 **Sun Java System Application Server 9.1 관리 설명서**를 참조하십시오.
로드 밸런서 구성	■ Web Server가 설치되어 있는지 확인합니다. ■ 로드 밸런서 플러그인이 Web Server에 설치되어 있는지 확인합니다. ■ 패치 확인이 비활성화되어 있는지 확인합니다. ■ `KeepAliveQuery` 매개 변수 값을 줄입니다. 값이 작을수록 부하가 적은 시스템의 대기 시간이 단축됩니다. 값이 클수록 부하가 높은 시스템의 처리량이 증가합니다. 자세한 내용은 **Sun Java System Application Server 9.1 Performance Tuning Guide**의 "Keep Alive"를 참조하십시오.

표 4–1 확인 목록 *(계속)*

구성 요소/기능	설명
Java 가상 머신 구성	■ 초기 최소 힙 및 최대 힙 크기를 각 인스턴스에 대해 1GB 이상의 동일한 값으로 설정합니다. ■ 자세한 내용은 Java Hotspot VM 옵션을 참조하십시오. ■ 여러 Application Server 인스턴스를 실행할 경우 프로세서 집합을 만들어 Application Server를 바인딩하는 것이 좋습니다. 이 방법은 CMS 컬렉터를 사용하여 이전 생성을 스윕하는 경우 유용합니다.
로드 밸런서의 시간 초과 구성	■ Response-time-out-in-seconds - Application Server 인스턴스가 비정상임을 선언할 때까지 로드 밸런서가 대기하는 시간입니다. 응용 프로그램의 응답 시간에 따라 이 값을 설정합니다. 너무 크게 설정하면 Application Server가 비정상으로 표시될 때까지 Web Server와 로드 밸런서 플러그인이 오래 대기합니다. 너무 작게 설정하여 Application Server의 응답 시간이 이 임계값을 넘어서면 인스턴스가 비정상으로 잘못 표시됩니다. ■ Interval-in-seconds - 비정상 인스턴스가 정상 상태로 돌아갔는지 확인할 때까지의 시간(초)입니다. 너무 작은 값은 로드 밸런서 플러그인에서 Application Server 인스턴스로 추가 트래픽을 발생시키고 너무 큰 값은 정상으로 전환된 인스턴스에 대한 요청 라우팅을 지연시킵니다. ■ Timeout-in-seconds - 상태 검사 요청에 대한 응답을 얻을 때까지의 기간입니다. 클러스터에 있는 시스템 간 트래픽에 따라 이 값을 조정하여 상태 검사가 성공하는지 확인합니다. 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 5 장, “HTTP 로드 균형 조정 구성”을 참조하십시오.

표 4–1 확인 목록 (계속)

구성 요소/기능	설명
HADB의 시간 초과 구성	■ sql_client_timeout - 유휴 클라이언트에 대한 SQLSUB의 대기 시간입니다. 예를 들어 로그온한 클라이언트에서 요청을 보낸 다음 사용자 입력을 기다리는데, 30분 이상 유휴 상태인 클라이언트는 사용 불능으로 간주되어 세션이 종료됩니다. 너무 작은 값을 설정하면 SQL 세션이 조기에 종료될 수 있습니다. 너무 큰 값을 설정하면 유휴 상태가 아니라 종료된 SQL 세션이 자원을 점유합니다. 이로 인해 다른 SQL 클라이언트가 로그온하지 못할 수 있습니다. 이 변수를 조정할 경우 `nsessions`의 설정도 고려합니다. HADB JDBC 연결 풀 steady-pool-size가 max-pool-size보다 크면 idle-timeout-in-seconds를 sql_client_timeout보다 작게 설정할 수 있으므로 HADB에서 연결을 닫기 전에 Application Server 자체에서 연결을 닫을 수 있습니다. 기본값은 1800초입니다. ■ lock_timeout - 데이터에 액세스할 때까지 트랜잭션이 대기하는 최대 시간(밀리초)입니다. 초과되면 트랜잭션에서 "트랜잭션 시간 초과" 오류 메시지를 생성합니다.이러한 시간 초과는 다른 트랜잭션(교착 상태)에서 점유한 잠금을 기다리는 트랜잭션에 의해 발생하며 높은 서버 부하를 발생시킵니다. 이 값을 500밀리초 미만으로 설정하지 마십시오. 서버 로그에 "트랜잭션 시간 초과" 메시지가 표시될 경우 이 값을 늘립니다. HADB의 JDBC 연결 풀에 다음과 같이 등록 정보를 추가하여 잠금 시간 초과 값을 설정합니다. `<property name=lockTimeout value="x"\>`기본값은 5000밀리초입니다. ■ Querytimeout - 쿼리가 실행될 때까지 HADB가 대기하는 최대 시간(밀리초)입니다. 서버 로그에 쿼리 시간 초과를 나타내는 예외가 지속적으로 나타날 경우 이 값을 늘려 보십시오. HADB의 JDBC 연결 풀에 다음 등록 정보를 추가하여 값을 설정합니다. `<property name=QueryTimeout value="x"\>`기본값은 30초입니다. ■ loginTimeout - HADB에 로그인할 때까지 클라이언트가 대기하는 최대 시간(초)입니다. HADB의 JDBC 연결 풀에 다음 등록 정보를 추가하여 값을 설정합니다. `<property name=loginTimeout value="x"\>`기본값은 10초입니다. ■ MaxTransIdle - 클라이언트에 응답을 보내고 다음 요청을 받을 때까지 트랜잭션이 유휴 상태일 수 있는 최대 시간(밀리초)입니다. HADB의 JDBC 연결 풀에 다음 등록 정보를 추가하여 값을 변경할 수 있습니다. `<property name=maxtransIdle value="x"\>`기본값은 40초입니다. 자세한 내용은 **Sun Java System Application Server Performance Tuning Guide**를 참조하십시오.

표 4–1 확인 목록 *(계속)*

구성 요소/기능	설명
Application Server의 시간 초과 구성	■ Max-wait-time-millis - 예외가 발생하기 전까지 풀에서 연결을 위해 대기하는 시간입니다. 기본값은 6초입니다. 지속되는 데이터의 크기가 50KB를 초과하는 고부하 시스템의 경우 이 값을 변경하는 것이 좋습니다. ■ Cache-idle-timeout-in-seconds - 비활성화되기 전에 EJB가 캐시에서 유휴 상태일 수 있는 시간입니다. Entity Bean과 Stateful Session Bean에만 적용됩니다. ■ Removal-timeout-in-seconds - EJB가 비활성화 상태(백업 저장소에서 유휴 상태)로 유지되는 시간입니다. 기본값은 60분입니다. 이 값은 SFSB 페일오버에 필요한 값을 기초로 조정합니다. HADB의 JDBC 연결 풀 설정 max-wait-time-in-millis에 주의하여 이 값을 모두 조정합니다. 자세한 내용은 **Sun Java System Application Server 9.1 고가용성 관리 설명서**의 "JDBC 연결 풀 구성"을 참조하십시오.
VM 가비지 모음(GC) 조정	가비지 모음이 4초 이상 일시 중지되면 HADB에 세션 상태를 지속하는 데 간헐적으로 문제가 발생할 수 있습니다. 이 문제를 방지하려면 VM 힙을 조정합니다. 데이터 지속 실패가 한 번도 허용되지 않거나 시스템이 완전히 로드되지 않은 경우에는 CMS 컬렉터나 처리량 컬렉터를 사용합니다. 컬렉터는 다음을 추가하여 활성화할 수 있습니다. `<jvm-options>-XX:+UseConcMarkSweepGC</jvm-options>` 이 옵션은 처리량을 감소시킬 수 있습니다.

색인

확

활