EJB 어플리케이션의 성능 모니터링 기법
(Performance Monitoring Techniques for EJB Applications)

나 학 청 † 김 수 동 ††
(Hak-Chung Na) (Soo-Dong Kim)

요 약 J2EE(Java 2 Enterprise Edition)의 등장으로 J2EE의 모델에 따르는 엔터프라이즈 어플리케이션의 개발이 이루어지고 있다. 이것은 J2EE의 핵심 기술 요소인 Enterprise JavaBeans(EJB)의 컴포넌트 모델이 분산 객체 어플리케이션의 개발을 간단하게 해주기 때문이다. EJB 어플리케이션은 컴포넌트 지향의 객체 트랜잭션 미들웨어를 사용하여 구현되며, 많은 어플리케이션이 분산 트랜잭션을 이용한다. 이러한 특징은 EJB 기술을 각광받게 하는 요인이 되었고, EJB 기반의 어플리케이션 개발에 관한 연구가 활발하게 이루어지게 하였다. 그러나 아직은 EJB 어플리케이션 운영 상태에서 성능을 측정하기 위한 기법에 대한 연구가 미흡하다.

본 논문에서는 운영 상태의 EJB 어플리케이션의 성능을 모니터링할 수 있는 기법을 제안한다. 우선 어플리케이션의 서비스를 위한 워크플로우를 살펴보고, 내부 작업을 여러 요소들로 분류한다. 제안된 기법은 분류된 여러 요소들 중 성능 요소들의 측정을 제공한다. 또한, 한 워크플로우 동안 발생하는 생명주기에 관련된 빈의 상태 변화와 빈에서의 처리시간, 자원 사용률과 같은 성능 정보를 추출하여 모니터링할 수 있다.

키워드 : EJB 어플리케이션, EJB 서버, 어플리케이션 측정, 모니터링 기법, 부하 측정

Abstract *Due to the emersion of J2EE (Java 2, Enterprise Edition), many enterprises inside and outside of the country have been developing the enterprise applications appropriate to the J2EE model. With the help of the component model of Enterprise JavaBeans (EJB) which is the J2EE core technology, we can develop the distributed object applications quite simple. EJB application can be implemented by using the component-oriented object transaction middleware and the most applications utilize the distributed transaction. Due to these characteristics, EJB technology became popular and then the study for EJB based application has been done quite actively. However, the research of techniques for the performance monitoring during run-time of the EJB applications has not been done enough.*

In this paper, we propose the techniques for monitoring the performance of EJB Application on the run time. First, we explore the workflow for the EJB application service and classify the internal operation into several elements. The proposed techniques provide monitoring the performance elements between the classified elements. We can also monitor by extracting the performance information like state transition and process time of the bean which is related to the lifetime occurred during one workflow, and the resource utilization rate.

Key words : Performance Measure, EJB Application, EJB Server, Monitoring Technique, Load Balancing

1. 서 론

1.1 동기 및 배경

1998년 SUN 사에 의해 EJB 명세서가 발표된 이후, 현재 국내·외 수많은 기업들이 EJB 기반의 어플리케이션을 개발하고 있다. 이것은 EJB에서의 컴포넌트 모델이 분산 객체 어플리케이션의 개발을 간단하게 해주기 때문이다. 분산 객체 어플리케이션은 트랜잭션적이며 확장성이 좋으며 이식 가능하다.

EJB 서버는 트랜잭션, 보안, 데이타베이스 연동 등과 같은 미들웨어 서비스를 제공하며, 클라이언트 프로그램

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 비 회 원 : 숭실대학교 컴퓨터학과
hchna@otlab.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.soongsil.ac.kr
논문접수 : 2002년 11월 16일
심사완료 : 2003년 3월 6일

과 EJB 어플리케이션 사이에서 메시지를 중재하여 외부로부터 직접적으로 빈을 접근할 수 없도록 한다. 또한 EJB 클라이언트 프로그램에게 트랜잭션 처리와 같은 EJB 서버내의 작업들은 감추어 어플리케이션 개발의 복잡도를 줄인다. 이 점은 EJB 개발자가 어플리케이션의 비즈니스 로직에 집중할 수 있도록 만든다[1].

이러한 특성은 EJB 기술을 선호하게 하는 요인이 되었고, EJB 기반의 어플리케이션 개발을 활발하게 하였다. 이에 따라 구축된 어플리케이션의 성능을 측정하는 것이 요구되었다. EJB 어플리케이션의 성능 측정을 위해서 EJB 서버를 만드는 회사가 제공하는 모니터링 도구나 그 EJB 서버와 함께 사용할 수 있는 써드파티 도구들을 이용한다. 이러한 도구들은 성능 측정을 위해 적용하고자 하는 성능 메트릭과 관계없이 사용되며, EJB 어플리케이션을 구축하기 위해 사용하는 EJB 서버에 종속적이다.

그러나 구축된 EJB 어플리케이션의 성능 측정에는 여러 가지 요인들이 존재하고 적용할 수 있는 메트릭이 다수 존재한다[2]. 본 논문에서는 미들웨어에 독립적이며, EJB 어플리케이션의 여러 가지 메트릭들을 적용할 수 있는 성능 측정 기법을 제시한다.

1.2 논문의 범위 및 구성

EJB는 J2EE의 핵심 기술로서 컴포넌트 기반의 분산 컴퓨팅을 위한 아키텍처다. EJB 서비스는 트랜잭션적인 컴포넌트와 분산 트랜잭션, 메시징과 비동기적 작업 관리, 보안과 인증 등을 포함한다. 이러한 서비스는 미들웨어인 EJB 서버에 의해서 지원되며, EJB 어플리케이션은 미들웨어를 이용하여 구현된다[3]. 본 논문에서는 이러한 EJB 어플리케이션의 특성에 따른 성능 모니터링 기법을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 어플리케이션 성능 모니터링 기법에 관한 기존 연구를 살펴보고, 3장에서는 EJB 어플리케이션의 성능 측정을 위한 스파이 에이전트 기법을 소개하며, 4장에서는 EJB 서버 측에서 사용할 수 있는 자바 프로파일링 기법을 제안한다. 5장에서는 앞 장에서 제안된 성능 측정 기법들을 사용하여 획득된 정보를 분석하는 기법을 설명하며, 6장과 7장에서는 제안된 기법들을 이용하여 EJBPerfMon이라는 성능 모니터링 시스템을 설계, 구현한다. 8장에서는 본 논문에서 제안한 성능 측정 기법들을 평가하며, 9장에서는 결론을 맺는다.

2. 관련 연구

2.1 EJB 개요

EJB는 재사용 가능한 컴포넌트라고 할 수 있는 빈이라는 용어를 사용한다. 빈은 세션(Session) 빈, 엔티티(Entity) 빈, 메시지 드리븐(Message-Driven) 빈의 세 가지 유형이 있으며, 각 빈은 어플리케이션에서 가장 기본적인 단위로서 성능적 관점으로 중요하게 다루어진다. 세션 빈은 비즈니스 프로세스를 모델링하며, 영구적 데이타를 나타내지 않는다. 즉, 세션 빈은 데이타베이스 안의 공유된 데이타나 다른 엔터프라이즈 자바 빈들과 같은 자원을 접근할 수 있지만, 기본적으로 비즈니스 로직을 구현한다. 엔티티 빈은 비즈니스 데이타를 재현한다. 즉, 엔티티 빈은 데이타베이스 안의 영구적인 데이타를 재현한다. 메시지 드리븐 빈은 세션 빈과 비슷하지만 비 동기적으로 메시지를 수신하여 처리한다[4].

클라이언트 프로그램은 빈 인스턴스를 Home Interface(원격 또는 로컬)와 Component Interface(원격 또는 로컬)를 통해 접근한다. 로컬과 원격 인터페이스의 차이는 클라이언트 프로그램이 같은 가상 머신(Virtual Machine) 안에 존재하는가의 여부에 따라 사용된다는 점이다. 클라이언트 프로그램은 JNDI(Java Naming and Directory Interface)를 이용하여 Home 객체를 얻고, 획득된 Home 객체를 통해 EJBObject를 생성하여 빈 인스턴스에 접근한다[5].

2.2 기존의 성능 측정 도구

EJB 어플리케이션은 BEA의 WebLogic Server와 IBM의 WebSphere Application Server, Oracle Application Server등과 같은 다양한 미들웨어를 사용하여 개발된다. 개발된 어플리케이션은 각각의 제품들에 의해 성능 측정 도구나 써드파티로서 제공된 측정 도구를 이용하여 어플리케이션의 성능을 측정할 수 있다. 그러나, 이 도구들은 서버에서의 자원 사용률이나 빈의 서버 내 배치된 개수, JNDI와 JMS, JDBC와 같은 서버 내의 자원 사용에 따른 현황 보고만을 하며, 빈 메소드의 응답시간 만을 외부에서 측정을 할 수 있도록 한다[6,7].

기존의 여러 성능 모델이나 메트릭들은 EJB 어플리케이션에 대한 성능 요소를 말하고 있다. EJB는 객체 기반이므로 기존의 객체 관련 메트릭[8]처럼 객체의 메시지 흐름에서 발생하는 여러 요소도 중요한 성능 요소이다. 또한, Liu의 Layered Queueing Models(LQM)[9]과 Lladó의 접근[10]은 EJB 어플리케이션에서 여러 가지 성능요소들을 말하고 있다. 그러나, 기존의 모니터링 도구들은 EJB 어플리케이션의 운영 시간에 성능요소들에 대한 측정 기능을 충분히 제공하지 않고 있다.

EJB 어플리케이션은 여러 개의 빈들과 객체들과 한

개 또는 여러 개의 서버로 이루어진다. 따라서 운영상태에서 EJB 어플리케이션의 성능을 측정할 때, 클라이언트의 서비스 처리를 서버와 빈들의 요소들로 분할할 필요가 있다. 이러한 서비스 처리의 분할 측정이 이루어져야 각 요소들의 성능이나 병목 지점을 파악할 수 있다.

기존의 성능 측정 도구들 - 미들웨어 각 회사들이 제공하는 도구들이나 Empirix사의 Bean-Test와 같은 써드파티 제품들 - 은 성능 측정이 자동화되어 있다는 장점이 있지만, 특정 미들웨어와 성능 측정에 적용되는 메트릭이 종속적이다. 따라서, 본 논문은 성능을 측정하고자 하는 어플리케이션에 독립적이며, 범용적으로 사용될 수 있는 성능 측정을 위한 기법을 제시하고자 한다.

2.3 자바 가상 기계 프로파일러 인터페이스(JVMPI)[1]

Java 2 SDK, Standard Edition은 JVMPI를 제공한다. JVMPI는 자바 가상 머신(JVM)을 프로파일링하기 위한 인터페이스로서, 프로파일러 에이전트를 만들 수 있도록 한다. 아직은 표준은 아니지만, J2SE에 의해 제공되고 있으므로 여러 플랫폼에서 JVM을 프로파일링을 하기 위해 이 인터페이스를 이용할 수 있다.

JVMPI는 Classic VM과 Java 2 Client VM, Java HotSpot 서버 VM 2.0에 구현되어 있으며, JVM과 내부 프로세스 프로파일러 에이전트 사이의 함수 호출 인터페이스로서 두 가지 방법을 제공한다. 하나는 가상 머신이 시스템 수행 중 발생하는 이벤트들을 프로파일러에게 알리는 것이며, 다른 하나는 프로파일러 에이전트가 JVMPI를 통해 더 많은 정보를 요청하고 제어하는 방법이다. 그림 1은 두 종류의 함수 호출 인터페이스를 보여주고 있다.

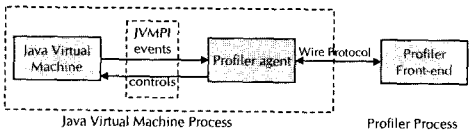

그림 1 JVMPI의 함수 호출 인터페이스

프로파일러 front-end는 프로파일러 에이전트와 같은 프로세스에 존재하거나 다른 프로세스에 존재할 수 있다. 이것은 같은 머신 상에서 다른 프로세스에 존재하거나, 네트워크를 통해 연결된 원격 머신 상에서 위치할 수도 있다. JVMPI는 표준 와이어 프로토콜을 명시하지 않는다. 따라서 사용하는 환경에 따라 적합한 와이어 프로토콜을 설계한다.

JVMPI는 서버에서 구동되고 있는 가상 머신과 통신하는 프로파일러를 만들 수 있도록 하는 저 레벨의 메커니즘을 제공한다. 즉, JVMPI는 서버상의 가상 머신과의 통신을 위한 API를 제공하는데, 이 API를 이용하여 가상 머신상에서 발생하는 이벤트와 정보를 획득할 수 있다.

3. EJB 서버에서의 스파이 에이전트 (Spy Agent)

3.1 엔터프라이즈 자바 빈

실행 시간에 빈 레벨에서의 성능측정을 위한 가장 쉬운 방법은 스파이 코드를 사용하는 방법이다. 이것은 빈의 종류에 따른 콜백(Call-Back) 메소드나 비즈니스 메소드들 안에 스파이 코드를 삽입하여 원하는 정보를 얻는 것이다. 즉, 빈의 종류에 따른 생명 주기와 메소드 호출 전파를 파악하고, 이러한 메시지에 스파이 에이전트를 삽입하여 빈의 상태 변화에 따른 이벤트를 측정할 수 있다. 본 논문에서는 세션 빈에 대해서 스파이 에이전트 기법을 설명하겠다.

그림 2은 상태 유지 세션 빈(SFSB) 의 생명 주기를 보여 주고 있다. 빈은 상황에 따라 ejbCreate <METHOD>(…)나 ejbActivate(…), ejbPassivate(…) 등의 콜백 메소드들이 컨테이너에 의해 호출되며, 클라이언트 프로그램의 비즈니스 메소드 호출에 대해 빈의 구현클래스의 비즈니스 메소드 호출이 이루어진다.

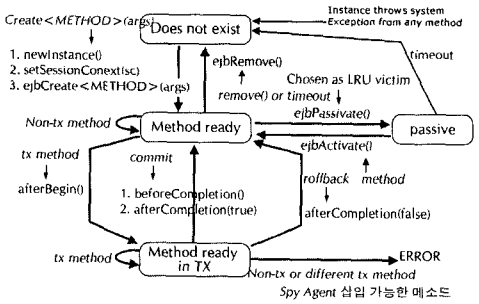

그림 2 상태 유지 세션 빈의 생명 주기

표 1은 상태 유지 세션 빈에 대해서 컨테이너에 의해 호출되는 콜백 메소드와 스파이 에이전트를 삽입하였을 때 획득될 수 있는 성능 정보를 기술한 것이다.

표 1 세션 빈의 콜백 메소드와 관련된 측정 정보

Method	SFSB
setSessionContext()	빈의 컨텍스트에 대한 저장이나 질의 시간을 측정.
ejbCreate…(…)	전달된 매개변수를 이용한 빈의 초기화 시간을 측정
ejbPassivate()	빈의 Passivation 시작시간 측정.
ejbActivate()	빈의 Activation 시작 시간 측정
ejbRemove()	빈 삭제 준비시간 측정

setSessionContext() 는 Session Context와 세션 빈을 관련지으며, 현재 트랜잭션적인 상태와 보안 상태 등에 대한 컨텍스트를 질의할 수 있도록 한다. 따라서 이 함수에 스파이 에이전트를 삽입함으로써 세션 빈의 컨텍스트에 대한 저장 및 질의시간을 측정할 수 있다. 또한 세션 빈을 초기화하는 메소드 ejbCreate…(…)에서는 빈의 초기화 시간을 측정할 수 있다. ejbPassivate(), ejbActivate()는 빈의 활성화와 비활성화 동작 시, 우선적으로 호출되는 메소드이다. 따라서 이 메소드에서는 활성화와 비활성화의 시작시간을 측정할 수 있다.

세션 빈은 이러한 콜백 메소드 외에 비즈니스 메소드에도 스파이 에이전트를 삽입함으로써 성능 정보를 측정할 수 있다. 이 정보를 이용하여 로직 수행 시간과 비즈니스 메소드에서 호출되는 여러 메소드들의 응답시간, 주어진 시간 동안에 비즈니스 메소드의 호출 빈도수를 파악할 수 있다.

프로그램 1 는 컴퓨터 부품 판매를 위한 전자 상거래에서 주문을 위한 세션 빈의 구현 클래스인 CartBean에 스파이 에이전트를 사용한 예이다. 객체 trace는 어플리케이션 내의 이벤트를 추적, 측정하는 클래스 Trace의 인스턴스이다. 실행시간에 어플리케이션의 트랜잭션을 측정하기 위해 각 이벤트에 대한 스파이 코드를 삽입함으로써 원하는 이벤트를 측정하는 것을 보여주고 있다. 이러한 정보를 이용하여 클라이언트의 호출 시작시간으로부터 경과시간과 각 메시지에 따른 응답시간, 실행시간 등의 중요한 정보를 획득할 수 있다.

3.2 스파이 에이전트의 설계 기법

EJB 어플리케이션의 성능측정을 위해 스파이 에이전트를 사용하는 경우, 각 유형의 빈에 대해서 콜백 메소드 안에 스파이 에이전트를 삽입하여 성능 정보를 측정하고 수집할 수 있다. 그러나 모든 빈에 대해서 스파이 에이전트를 삽입하는 경우, 코드의 중복이 발생하여 비효율적이다. 이러한 문제는 설계 단계에서 상속관계를 이용하여 해결할 수 있다. 그림 3은 세션 빈과 엔티티 빈에 스파이 에이전트를 사용하기 위해 상속 장치를 이용하여 설계한 것이다.

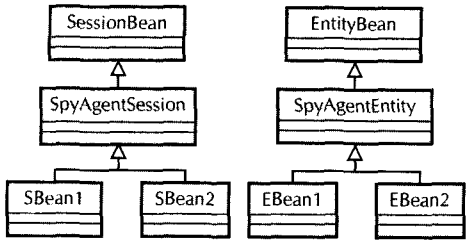

그림 3 상속을 이용한 스파이 에이전트 사용

세션 빈과 엔티티 빈의 구현 클래스는 SessionBean 인터페이스와 EntityBean 인터페이스를 상속한다. "인터페이스 - 빈 구현클래스" 순서의 상속구조에서 "인터페이스 - 스파이 에이전트 클래스 - 빈 구현 클래스" 순서의 상속구조로 중간에 스파이 에이전트 클래스를 추가한다. 빈의 종류에 따라 생명주기가 다르기 때문에 각 빈의 유형별로 인터페이스를 상속받는 스파이 에이전트 클래스를 만들고 이를 각각의 빈이 상속을 받아 구현한다.

SpyAgentXXX 클래스는 상위 인터페이스인 SessionBean과 EntityBean을 구현하여 나중에 구현될 세션 빈과 엔티티 빈에서 스파이 에이전트를 사용한 성능 정보 수집을 용이하게 한다. 또한, 이 클래스는 SpyAgent의

프로그램 1 세션 빈의 스파이 에이전트 사용

```
public class CartBean implements SessionBean {
        public void ejbActivate() {
                trace.setEvent("ejbActivate()", "arrived time");
                ...
        }
        public void ejbCreate() throws CreateException {
                trace.setEvent("ejbCreate()", "Execution start time");
                ...
                trace.setEvent("ejbCreate()", "Execution end time");
        }
        public double getSubtotal() throws RemoteException {
                trace.setEvent("getSubtotal()", "Execution end time");
                ...
                trace.setEvent("getSubtotal()", "Execution end time");
}}
```

정적 인스턴스를 가지며, 각 메소드들을 구현함으로써 이벤트에 따른 성능 정보를 수집할 수 있으며, 이를 기반으로 생명주기에 따른 경과시간이나 비즈니스 메소드들의 실행 시간과 여러 콜백 메소드들의 처리시간을 획득할 수 있다.

4. EJB 서버에서의 자바 프로파일링 기법

4.1 EJB 어플리케이션의 프로파일링

자바 프로파일러 에이전트는 EJB 컨테이너가 실행되고 있는 JVM의 항목들을 볼 수 있다. 즉, 실행된 어플리케이션에서 모든 메소드들이 소비하는 시간을 볼 수 있으며, CPU 사용률이나 객체 할당에 대한 프로파일을 생성할 수 있다. 또한, 어플리케이션이 실행되고 있는 Java 힙들에 대해 현재 상태의 스냅샷(Snapshot)을 얻을 수 있다. 이 스냅샷은 어떤 클래스의 객체들이 얼마나 많이 배치되어 있으며, 그들을 누가 참조하고 있는 가와 같은 정보를 갖고 있다. 이 정보는 객체 할당이나 불필요한 메모리 사용문제 등을 파악할 수 있게 한다[12].

EJB 어플리케이션에 대한 프로파일링의 목적은 시스템의 어떤 부분이 대부분의 자원을 소비하는지를 파악하는 것이다. 따라서 좋은 EJB 어플리케이션 프로파일링 도구는 빈 메소드 들에 대한 호출 빈도수와 소비시간율, 메모리 사용율, 높은 호출 빈도를 갖는 빈 메소드들을 사용하는 빈 메소드 정보를 제공해야 한다. 또한, 실행시간 프로파일링과 메모리 사용 프로파일링, Flat 프로파일링의 기능이 요구된다. 본 논문에서는 메소드에 대한 실행시간과 메모리 사용 측정, Flat 프로파일링의 기능을 설계하기 위해 JVMPI를 이용할 것이다.

4.2 JVMPI를 이용한 프로파일러 설계 기법

어플리케이션을 구성하는 서버는 JVM에서 구동되고 있다. 서버가 구동되는 각각의 JVM에는 프로파일러 에이전트가 동작하며, 프로파일러 에이전트는 RMI를 통하여 프로파일러 클라이언트와 통신을 하며 성능 정보를 수집한다. 그림 4은 i 번째 EJB 서버에 대한 프로파일러의 구성을 보여주고 있다. 그림에서 번호는 프로파일링 클라이언트의 빈 메소드 실행 프로파일링을 수행하는 순서를 나타낸다.

EJB 서버는 JVM 위에서 구동된다. EJB 어플리케이션의 EJB 서버들에 프로파일러 에이전트와 빈에 대한 프로파일링된 정보를 수집하는 빈을 적재하여 프로파일러 클라이언트가 그 정보를 받을 수 있도록 한다. 프로파일러는 하나의 자바 클래스 ProfWorker와 프로파일링 엔진으로 구성된다. ProfWorker는 Java Native

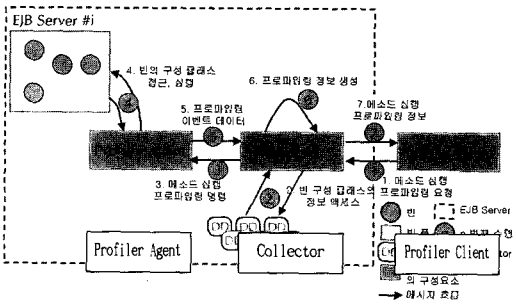

그림 4 i EJB 서버에 대한 프로파일러 구성

Interface (JNI)를 이용하여 프로파일링 엔진과 통신하며, Collector에게 프로파일링 엔진에 대한 핸들을 제공한다. Collector는 세션 빈으로 구성되며, 프로파일러 클라이언트로부터 프로파일링 요청을 받고 프로파일링 결과를 프로파일러 클라이언트에게 전달한다.

Collector는 실행시간에 프로파일링하려는 메소드와 그 메소드를 갖는 클래스의 이름, 메소드의 매개변수들, 프로파일링 옵션을 프로파일러 에이전트로 보내어 프로파일링을 한다. 또한 Collector는 프로파일링 하려는 빈 메소드에 대해, 메소드를 갖는 클래스의 이름을 - EJB 서버에 배치된 클래스 - 획득하고, 빈 메소드를 실행하기 위한 임의의 메인 클래스를 만든다. 빈은 EJB 서버에 배치될 때, 빈의 구성 클래스들은 EJB 서버 제품에 따른 Naming 규칙을 따른다. 따라서, 그 규칙에 따라 구성 클래스들에 대한 적절한 클래스 이름을 획득할 수 있다. 이 클래스 이름을 사용하여 실행할 수 있다.

5. 측정된 성능 정보의 해석 분석

5.1 측정 가능한 성능 정보

본 논문에서 제시되는 3가지 성능 측정 기법을 이용하여 EJB 어플리케이션의 클라이언트 측에서의 서비스 요청으로부터 서버 측까지 발생되는 워크플로우의 흐름들에 대한 성능 정보를 얻을 수 있다. 이 성능 정보는 EJB 어플리케이션의 서비스 시간과 워크 플로우를 구성하는 각각의 빈들의 서비스 시간, 자체 로직 시간, 빈 메소드들의 시간 소비율을 포함한다.

표 2은 기법을 이용하여 측정할 수 있는 정보를 보여주고 있다. EJB 클라이언트에서의 성능 측정 기법을 이용하여 EJB 어플리케이션의 서비스 시간을 측정할 수 있다. 스파이 에이전트 기법과 자바 프로파일링 기법을 이용하여 워크플로우에 참여하는 각각의 빈들에 대해 응답시간과 메소드 실행시간을 파악하며, EJB 어플리케이션의 서비스 시간에 대한 소비율을 측정된 정보들을

기반으로 분석할 수 있다.

5.2 성능 정보 분석 방법

측정된 성능 정보를 이용하여 EJB 어플리케이션의 서비스 시간을 소비하는 각 빈들로부터 성능이상(부하와 같은)이 발견되는 빈을 찾아낼 수 있다. 이것은 서로 다른 서버상에 위치하는 빈들과 빈 클라이언트 사이에 발생하는 네트워크 상의 가변적인 요소들을 추출하고, EJB 어플리케이션의 성능 요소들을 세분화함으로써 가능하다. 성능 정보를 분석하여 다음과 같이 성능 향상 방법을 획득할 수 있다.

우선, 하나의 빈 메소드에 대한 로직과 의존 되는 빈의 성능 정보를 이용하여 현재 빈 메소드에 대해서 성능 향상 지점을 판단할 수 있다. 또한, 로직을 최적화할 것인지 의존되는 빈의 성능을 향상시킬 것인 지를 결정하여 성능을 향상시킬 수 있다.

둘째로 워크플로우에 참여하는 각 빈의 실행시간 정보 획득은 EJB 어플리케이션의 워크플로우를 구성하는 각각의 빈들에 대해서 다른 시간 요소들과 비교, 분석할 수 있게 한다. 또한 성능 향상지점을 판단할 수 있다.

셋째로 어떤 빈이 성능이상이 발견되었고, 그 빈에 대해 바인딩 시간을 많이 소비한다면, 어플리케이션의 성능향상을 이루기 위해 빈이 운영되는 환경을 조정할 수 있다.

넷째로 워크플로우에 참여하는 여러 빈들 중에 어떤 빈이 부하가 높고, 그 빈이 높은 데이타베이스 사용율을 갖고 있다면, 불필요한 데이타베이스로의 액세스 횟수를 억제하고 데이타베이스 측에서 성능 최적화를 한다. 또한, 빈을 위한 데이타베이스 연결 풀링이나 자원 할당을 조정하여 해결할 수 있다.

그리고 워크플로우에서 특정 빈에서의 대기시간이 높고, 그 빈이 낮은 데이타베이스 사용율을 갖는다면, 빈에서 사용하는 서버 측 자원들(데이타베이스 연결 풀링과 같은 자원, 빈을 위해 할당된 자원들)을 조정하거나 그 빈의 로직 최적화로 성능을 향상시킬 수 있다.

6. EJBPerfMon 시스템 설계

6.1 기능적 요구 사항

EJBperfMon 시스템의 기능적 요구사항은 앞 장에서 언급한 여러 측정기법들을 통해 나타나 있다. 이 기능은 크게 두 가지로 나타난다. 첫 째는 EJB 어플리케이션을 구성하는 EJB 서버의 자원 사용율을 파악하는 것이다. 두 째는 워크플로우에 대한 성능 측정이다. 워크플로우에 대한 성능 측정은 전체 또는 단계별 응답시간, 메소드 호출 빈도수, 메소드의 메모리 점유율을 파악하는 것을 말한다. 워크플로우는 메소드의 호출로서 이루어지기 때문에 단계별 응답시간은 워크플로우를 구성하는 메소드들의 소비시간으로 볼 수 있다. 메소드 호출 빈도수, 메소드의 메모리 점유율, 메소드 시간 소비율로 다음과 같은 정보를 파악할 수 있다.

6.2 EJBPerfMon의 아키텍처

EJB 어플리케이션은 하나 또는 두 개 이상의 EJB 서버를 사용하여 구축된다. 하나의 EJB 서버를 이용하여 구축된 EJB 어플리케이션을 위한 성능 모니터는 가장 간단한 방법으로 구축된다. 두 개 이상의 다중 EJB 서버를 이용하여 구축된 EJB 어플리케이션은 좀 더 복잡한 방법이 요구된다.

표 3 워크플로우의 성능 요소 별 획득 정보

	획득정보
메소드 호출 빈도수	자주 호출되는 메소드들과 자주 호출되는 메소드들을 호출하는 메소드들에 대한 정보.
메소드 시간 소비율	워크플로우에서 대부분의 시간을 소비하는 메소드들에 대한 정보.
메소드의 메모리 점유율	많은 메모리를 점유하는 메소드들에 대한 정보.

표 2 측정되는 정보

	설 명
메소드 레벨에서의 객체생성 및 메소드 호출 정보	하나의 메소드가 실행 되었을 때, 생성되는 객체와 호출되는 메소드들에 대한 정보를 획득.
메소드 실행시간	하나의 메소드가 호출되었을 때, 메소드의 진입 지점으로부터 메소드가 실행이 종료되는 시간 간격.
빈의 바인딩 시간	어떤 빈에 대한 Home, EJBobject의 바인딩 시간.
빈 메소드 자체실행시간	어떤 빈 메소드에 사용되는 다른 빈들의 서비스 시간을 제외한 자체 실행 시간.
활성화 시간	빈 인스턴스의 활성화 시간.
빈 메소드 응답 시간	빈 클라이언트에서의 빈 메소드가 실행되고 그 결과를 반환 받는 시간 간격.
EJB 어플리케이션의 서비스 시간	어플리케이션의 클라이언트가 하나의 요청을 처리하기 위해 EJB 어플리케이션으로부터 서비스 받는 시간

6.2.1 단일 서버 어플리케이션의 성능 모니터

단일 서버를 이용하여 구축된 EJB 어플리케이션의 경우, 구동되는 EJB 서버에 EJBPerfMon이 함께 구동된다. EJBPerfMon은 EJBPerfMon Server와 Collector, 프로파일러 에이전트(Profiler Agent)로 구성된다. 프로파일러 에이전트는 ProfWorker(자바 어플리케이션)와 ProfilerEngine(.DLL)로 구성된다. Collector는 프로파일러 에이전트에게 프로파일링 요청하며, 에이전트는 Profiler를 사용하여 프로파일링한다. ProfWorker는 프로파일링 요청을 Collector로부터 받고, Profiler의 결과를 Collector에게 보낸다. 그림 5는 EJBPerfMon이 모니터링을 수행하는 것을 보여주고 있다.

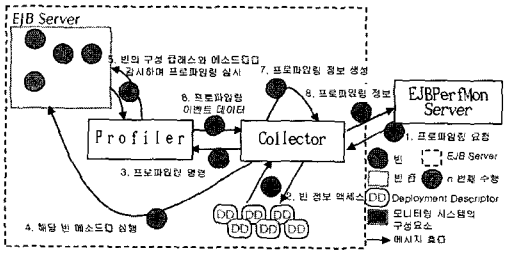

그림 5 단일 서버 EJB 어플리케이션의 성능 모니터링 수행

단일 서버 EJB 어플리케이션 모니터링은 8 단계에 걸쳐 수행된다. (1) EJBPerfMon Server는 Collector에게 어플리케이션의 프로파일링을 요청한다. 이 때, EJBPerfMon Server는 프로파일링에 필요로 하는 정보로 프로파일링 유형(리소스 사용율, 워크플로우) 옵션, 메소드 이름과 인수등을 전달한다. (2) Collector는 EJB 어플리케이션을 구성하는 빈들의 정보를 디플로이먼트 디스크립터(Deployment Descriptor)로부터 획득한다. (3, 4) Collector는 획득된 정보를 이용하여 프로파일러 에이전트에게 프로파일링을 명령하고, 해당 빈 메소드를 호출, 실행한다. (5) 목표 빈과 메소드들을 감시하며, 프로파일링을 시작한다. (6, 7) 프로파일링 이벤트 데이타는 Collector로 전달되어 프로파일링 정보로 처리된다. (8) 프로파일링 정보는 EJBPerfMon Server로 보고한다.

6.2.2 다중 서버 어플리케이션의 성능 모니터

여러 개의 서버들을 이용하여 구축된 EJB 어플리케이션의 경우, 구동되는 EJB 서버들과 함께 EJBPerfMon이 구동된다. EJBPerfMon은 EJBPerfMon Server와 Collector, 프로파일러 에이전트(Profiler Agent)로 구성된다. 프로파일러 에이전트는 ProfWorker(Java App.)와 Profiler(.DLL)로 구성된다. 그림 6은 EJBPerfMon의 모니터링 수행을 보여주고 있다.

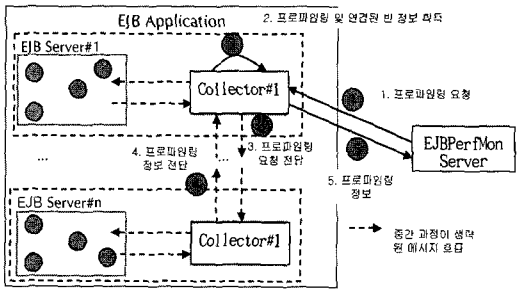

그림 6 다중 서버 EJB 어플리케이션의 성능 모니터링 수행

다중 서버 EJB 어플리케이션 모니터링은 5 단계에 걸쳐 수행되며, 포함되는 Collector의 수행 단계는 단일 서버의 경우와 동일하다. (1) EJBPerfMon Server는 Collector#1에게 어플리케이션의 프로파일링을 요청한다. 이때, EJBPerfMon Server는 프로파일링에 필요로 하는 정보로 프로파일링 유형(리소스 사용율, 워크플로우) 옵션, 메소드 이름과 인수를 전달한다. (2) Collector#1은 현재 서버에 대해 프로파일링하고, 연관된 빈이 있으면 그 빈에 대한 정보 획득한다. 이 때, Collector#1의 프로파일링은 단일 서버 EJB 어플리케이션 모니터링 수행순서 2~7을 실시한다. (3) 연관된 빈이 존재하면, 그 빈이 위치한 EJB Server와 연결된 Collector에게 프로파일링을 요청한다. 이 때, 프로파일링을 요청받은 Collector는 (2)와 (3)을 반복한다. (4) 프로파일링 요청을 전달 받은 Collector는 프로파일링 정보를 요청한 Collector에게 전달한다. (5) 수집된 프로파일링 정보를 EJBPerfMon Server로 보고한다.

6.3 클래스 다이어그램

그림 7은 EJBPerfMon의 정적인 구조를 보여주는 클래스 다이어그램이다. EJBPerfMon은 시간 측정을 위한 클래스들 STimerCollections, STimer, STimeData와 하나의 EJB 서버에 배치되어 EJB 서버 내의 측정된 성능 데이타를 수집하고 EJBPerfMon 서버로 전달하는 세션 빈 Collector, EJB 서버 내에 배치되어 있는 빈 정보에 대한 EJBInfo, EJB 서버를 모니터링하는 프로파일러 에이전트로 구성된 것을 볼 수 있다. 프로파일러 에이전트는 ProfWorker와 JVMPI를 이용하여 만들어진 프로파일링 엔진 Profiler로 구성되어 있다.

7. EJBPerfMon 시스템 구현

7.1 EJB 어플리케이션 모니터링 알고리즘

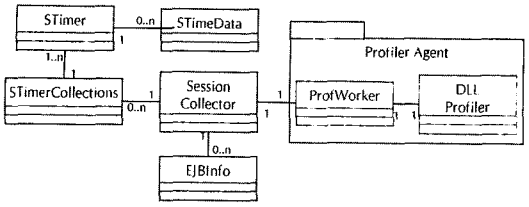

그림 7 EJBPerfMon의 클래스 다이어그램

EJB 어플리케이션은 EJB 서버 위에 구동되며, 기본적으로 자바 가상 머신 상에서 구동된다. EJBPerfMon의 구성요소인 Collector는 각 서버마다 배치되며, 서로 독립적으로 모니터링한다. Collector는 EJB 서버의 환경으로부터 EJB 빈에 대한 정보를 획득하고, 획득한 정보를 프로파일러 에이전트에 전달한다. 프로파일러 에이전트는 빈정보를 이용하여 EJB 서버 내부에서 수행되는 워크플로우를 모니터링한다. 그림 8은 프로파일러 에이전트에서 모니터링 알고리즘을 보여주고 있다.

```
/* isProfStart : 디폴트는 false이며, 워크플로우를 시작을 나타내는 플래그 */
/* isFirst : 디폴트는 true이며, 모니터링할 빈 메소드에 대한 진입여부를 나타내는 플래그. */
/* profTargetMethodName : 프로파일링할 목표 메소드 이름. */
/* methodNameTobeMonitors : 측정해야할 메소드 목록들. */
/* profTargetBeanName : 프로파일링할 목표 빈 이름. */
/* activationTime : 빈의 활성화 시간 */
/* startTime : 메소드 실행 시작시간 */
/* execTime : 메소드 실행 시간 */

static void notifyEvent(JVMPI_Event *event) {
    switch((event->event_type) & (isFirst == true) & (isProfStart == true)){
            case JVMPI_EVENT_CLASS_LOAD:
                빈에 필요한 클래스 정보 획득;
            case JVMPI_EVENT_METHOD_ENTRY:
                if(이벤트의 메소드이름 = profTargetMethodNam) {
                        현재 메소드의 startTime = getSystemTime();
                        isFirst = false;
                }
                for(int i = 0;i<methodNameTobeMonitors 개수;i++) {
                        if( (이벤트의 빈이름 = profTargetBeanName) &
                            (이벤트의 메소드이름 = methodNameTobeMonitors[i]) {
                            현재 메소드의 startTime = getSystemTime();
                            return;
                } }
                break;
    ... }
    switch((event->event_type) & (isFirst == false) & (isProfStart == true)) {
            case JVMPI_EVENT_METHOD_ENTRY:
                if(이벤트의 메소드이름 = profTargetMethodNam)
                        현재 메소드의 startTime = getSystemTime();
                for(int i = 0;i<methodNameTobeMonitors 개수;i++) {
                        if( (이벤트의 빈이름 = profTargetBeanName) &
                            (이벤트의 메소드이름 = methodNameTobeMonitors[i]) {
                            현재 메소드의 startTime = getSystemTime();
                            return;
                } }
                break;
            case JVMPI_EVENT_METHOD_EXIT:
                if(이벤트의 메소드이름 = profTargetMethodNam) {
                        startTime = 현재 메소드의 실행시작 시간;
                        현재 메소드의 execTime = getSystemTime() - startTime;
                        return;
                }
                for(int i = 0;i<methodNameTobeMonitors 개수;i++) {
                        if( (이벤트의 빈이름 = profTargetBeanName) &
                            (이벤트의 메소드이름 = methodNameTobeMonitors[i]) {
                            startTime = 현재 메소드의 실행시작 시간;
                            현재 메소드의 execTime = getSystemTime() - startTime;
                } }
                break;
            ...    // 필요한 여러 이벤트들을 등록.
} }
```

그림 8 EJB 어플리케이션 모니터링 알고리즘

7.2 적용사례

본 절에서는 구현된 EJBPerfMon를 이용하여 실제 EJB 어플리케이션의 성능측정에 적용해 본다. 성능을 측정하기 위한 목표 시스템은 트랜잭션 매우 많은 뱅킹(Banking) 어플리케이션이며, 대상 업무로서 은행의 여신 업무를 사용하여 성능을 측정해 본다. EJBPerfMon을 구동하기 위한 시스템 환경은 표 4와 같으며, 뱅킹 어플리케이션과 성능 측정을 위한 클라이언트 프로그램은 동일한 환경에서 구동된다.

표 4 시스템 환경

운영체제	Windows XP
자바 가상 머신	JDK 1.3.1
EJB 서버	WebLogic 6.1
데이타 베이스	Oracle 8i

뱅킹 어플리케이션의 S/W 구조는 그림 9과 같이 구성된다. 뱅킹 시스템은 하나의 마스터(Master) EJB 서버와 슬레이브(Slave) 서버가 관리하는 세 개의 슬레이브 EJB 서버들, 각 Managed EJB 서버에 의해 사용되는 세 개의 데이타 베이스들로 구성된다. 마스터 EJB 서버는 클라이언트의 요청을 받아들이고 비즈니스 로직을 처리하며, 컨트롤러 계층과 비즈니스 계층의 세션 빈들이 배치된다. 슬레이브 EJB 서버에는 데이타베이스와 통신하며, 데이타 접근 계층의 엔티티 빈들이 배치된다.

우선, EJBPerfMon은 모니터링할 EJB 서버 정보를 요청한다. 입력된 정보를 바탕으로 모니터링될 EJB서버를 구동시키고, 그 EJB 서버에 배치된 Collector를 획득한다. 두 번째로 빈과 빈 메소드에 대한 정보를 설정하여 해당 빈 메소드를 모니터링 한다. 그림 10은 EJBPerfMon이 모니터링할 EJB 서버를 설정하는 화면을 보여주고 있다. 이 EJB 서버에 대한 정보는 서버가 위치하고 있는 호스트 이름이나 IP, 모니터링할 EJB 서버의 제품 버전, 구동을 위한 클래스패스이다.

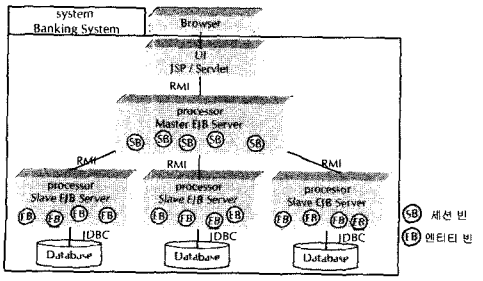

그림 9 뱅킹 어플리케이션의 S/W 구조

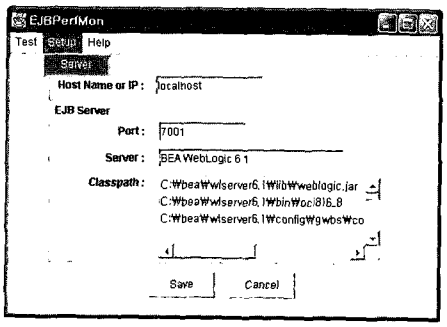

그림 10 EJB 서버 설정

그림 11은 모니터링할 빈과 빈 메소드에 대한 정보를 설정하는 화면이다. 빈에 대한 정보를 획득하기 위해 목적 빈의 .jar 파일의 이름이 필요하다. 또한, 모니터링을 위한 빈 메소드의 정보가 필요하다. 모니터링 결과에 대한 Running Type은 시간과 목적 빈 메소드의 실행시간에 따른 소비시간율에 대한 유형을 말한다.

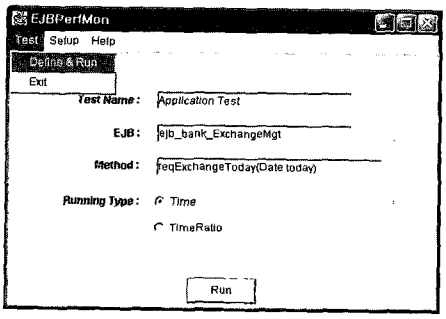

그림 11 EJBPerfMon의 빈 설정

그림 12은 설정된 빈 메소드에 대한 측정결과이다. EJB 빈에 대한 .jar파일에는 EJB 서버에 의해 생성된 여러 클래스들을 포함하고 있으므로 해당 빈의 .jar 파일의 이름과 빈 메소드 이름을 조합한 이름과 측정값을 사용하여 보고된다. 또한, 빈 메소드들의 호출관계는 트리구조로서 보여지며, 상위 노드는 하위 노드의 실행을 포함한다.

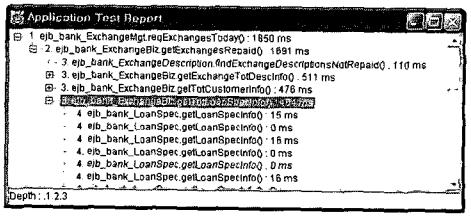

그림 12 EJBPerfMon의 측정 결과

8. 평 가

EJB 어플리케이션은 BEA의 WebLogic Server와 IBM의 WebSphere Application Server, Oracle Application Server등과 같은 다양한 미들웨어를 통해 개발된다. 이들 미들웨어 마다 여러 가지 측정도구들을 제공하지만, 서버에서의 자원 사용률이나 빈의 서버 내 배치된 개수, JNDI와 JMS, JDBC와 같은 서버 내의 자원 사용에 따른 현황 보고에 그치고 있다. 이것은 써드파티로서 제공되는 측정도구들 또한 마찬가지이다.

EJB 어플리케이션의 성능을 측정하기 위한 메트릭이 다수 존재한다. 특히, 워크플로우를 측정하는 메트릭을 위해 기존의 제공되는 도구들을 사용하기에는 역부족이다. 우리는 3, 4, 5 장을 통해서 세 가지 기법을 제시하였다. 제안된 기법들은 빈과 그들간의 메시지 호출, 사용되는 객체, EJB 서버 등과 같이 EJB 어플리케이션이 갖고 있는 여러 특징에 따라 객체와 빈 단위로 측정할 수 있도록 하였다. 또한, 이 기법들을 통해서 EJB 어플리케이션의 클라이언트 측으로부터 서버 측까지의 워크플로우의 모니터링이 가능하다.

다음의 표 5은 제안된 세가지 모니터링 기법과 기존의 기법을 6가지 항목에 대해 비교한 것이다. 기존의 기법들의 이름이 그 기법이 사용된 제품의 이름으로 나타냈다. 그리고, 써드파티 제품군들은 Wily Technology사의 Introscope와 Expirix사의 Bean-Test등을 포함한다.

빈 생명 주기에서 나타나는 이벤트들은 EJB 빈들의 성능 요소들 측정하기 위한 중요한 기본 자료이다. 따라서, 비교 항목은 빈 생명 주기의 이벤트를 포함해서 서버와 빈에서 객체 단위까지의 측정과, 워크플로우를 구성하는 각 빈들에 대한 측정을 대한 워크플로우 테스트로 구성된다.

WebLogic과 WebSphere, Oracle Application Server는 서버와 빈 단위로 자원 활용 모니터링을 제공한다. 또한, 서버내에서 빈의 활성화와 비활성화 등 생명 주기에 대한 횟수를 파악하여 간접적으로 측정 가능하다. 그러나 시간 활용율에 대해 파악할 수 없으며, 하나 이상의 서버에 대해서 CPU, 메모리 활용율과 같은 자원 활용율을 알 수 없다.

써드파티 제품군들은 제품 마다 약간의 차이가 있지만, 서버와 빈, 객체 단위에 대해서 여러 가지 측정을 제공하고 있다. 그러나, 빈의 생명 주기에 따른 성능요소와 빈 메소드가 호출되었을 때의 자원 활용율, 워크플로우를 구성하는 각 빈들에 대한 고려가 부족하다.

본 논문에서 제시한 세가지 기법들은 워크플로우 테스트를 통해 워크플로우에 참여하는 각 빈들과 빈 메소드들에 대한 시간, 자원 활용율에 대한 분석이 가능하다. 또한, 빈 메소드가 호출되었을 때, 사용되는 객체들에 대한 시간 및 자원활용율, 빈 생명 주기의 이벤트에 대한 분석이 가능하다. 이는 기법들이 어플리케이션, 서버, 빈 단위뿐만 아니라, 워크플로우를 구성하는 빈들에 대해 어플리케이션 성능에 영향을 주는 여러 요소들에 대한 정보를 획득하기 때문이다.

EJB 어플리케이션의 성능 메트릭은 어플리케이션 내의 서버와 빈뿐 아니라, 빈 메소드, 빈의 활성화, 사용되는 객체들에 대한 정보를 포함하고 있다. 따라서 메트릭에 필요로 하는 기본적인 정보를 제공함으로써 실행시간에 EJB 어플리케이션의 측정에 해당 메트릭을 적용할 수 있도록 했다. 또한, 빈 메소드 호출 메커니즘에 따라 발생하는 빈 인스턴스 활성화와 메시지 전파를 고려하여 워크플로우를 구성한 빈 메소드에 대한 정확한 성능 측정을 할 수 있었다. 획득된 성능 정보를 바탕으로 각각의 빈들에 성능이상의 발생지점을 명확하게 판

표 5 제안된 모니터링 기법과 기존의 기법 비교

운영 상태에서의 단위 별 자원 활용율, 시간 활용율 측정 지원						
	서버 단위	빈단위	객체 단위	빈 메소드	워크플로우 테스트	빈 생명 주기의 이벤트
WebLogic	△	○				△
WebSphere	△	○				△
Oracle Application Server	△	○				△
써드파티 제품군	○,▲	●	○,▲	●	▲	
EJB 클라이언트에서의 성능 데이타 측정 기법	●	●	▲	●		
스파이 에이전트 기법	●	●	○,●	○,●	○,●	▲
EJB 서버에서의 자바 프로파일링 기법	△,●	△,●	○,●	○,●	△,●	○,●

자원 활용율 지원((: 부분 지원, (: 지원), 시간 활용율 지원((: 부분 지원, (: 지원)

단할 수 있었다.

9. 맺음말

본 논문에서는 운영 상태의 EJB 어플리케이션의 성능을 모니터링할 수 있는 기법을 제안하였다. 제안된 기법은 어플리케이션의 서비스를 위한 워크플로우 동안 발생하는 생명주기에 관련된 빈의 상태 변화와 빈에서의 처리시간, 자원 사용률과 같은 성능 정보를 추출하여 모니터링한다. 이것은 주어진 워크플로우에 따른 EJB 어플리케이션의 클라이언트 계층에서의 응답시간으로부터 어플리케이션을 구성하는 서버, 빈들에 이르기까지 세부적인 요소를 반영할 수 있도록 한 것이다.

또한 사례연구를 통하여 제시된 모니터링 기법 및 성능 메트릭의 적용성과 효율성을 보였다. 8장의 시스템 구현을 통해서 구현된 시스템이 워크플로우를 모니터링하기 위한 최소한 정보만을 요구하는 것을 보였다. 이것은 EJB 빈이 제공되어 있을 때, 외부에 대해 블랙박스 형태로서 얻을 수 있는 최소한의 정보만으로도 충분히 모니터링이 가능한 기법임을 보여준다. EJB 어플리케이션의 시스템 운영자는 본 논문의 기법들을 이용하여 어플리케이션의 성능 측정을 통해 어플리케이션의 병목부분을 파악할 수 있다 또한, 분석된 정보는 어플리케이션을 구성하는 서버의 튜닝에 사용할 수 있다.

참 고 문 헌

[1] Halter S., Munroe S., *Enterprise Java Performance*, Prentice Hall PTR, Aug. 2000.

[2] Fenton N., Pfleeger S., *Software Metrics: A Rigorous & Practical Approach*, PWS Publishing Company, 1997.

[3] Java Community, *ECperf Specification, Sun Microsystems*, at URL : http://java.sun.com/j2ee/ecperf/download.html, May 29, 2001.

[4] Sun Microsystems, *Enterprise JavaBeans Specification, Version 2.0*, at URL : http://java. sun.com/products/ejb/docs.html, Aug. 14, 2001.

[5] Roman Ed., Mastering Enterprise JavaBeans, Second Edition, WILEY, 2002.

[6] Girdley M., et al, *J2EE Applications and BEA WebLogic Server*, Prentice Hall PTR, 2002.

[7] Adatia R., et al., *Professional EJB Volume 2*, Worx Press Ltd., 2001.

[8] 김철진, 조은숙, 김수동, "효율적인 객체지향 설계 및 성능 측정을 위한 정적/동적 메트릭", 한국정보과학회논문지(B), 제 25 권, 제 11 호, pp.1657- 1666, 1998, 11.

[9] Liu T., et al., "Layered Queueing Models for Enterprise JavaBean Applications", Fifth IEEE International Enterprise Distributed Object Computing Conference, Sept. 2001.

[10] Lladó C.M., Harrison P.G., "Performance evaluation of an enterprise javabean server implementation.", In: Proc. 2nd, Int. Workshop on Software and Performance (WOSP 2000, September 17-20, Ottawa, Canada), 2000.

[11] Sun Microsystems, Java Virtual Machine Profiler Interface (JVMPI), Apr. 13, 2002

[12] Steve W., Jeff K., *Java Platform Performance Strategies and Tactics*, Addison-Wesley, Jun., 2000.

나 학 청

2001년 순천향 대학교 전산학과 졸업. 2003년 2월 숭실대학교 컴퓨터학과 석사 졸업. 2003년 2월~현재 이비즈온㈜. 관심분야는 EJB 소프트웨어 성능 공학, 컴포넌트 개발 방법론.

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 1 호 참조