J2EE 기반의 엔터프라이즈 애플리케이션 성능 저하 요인에 대한 연구

김민규*, 최성운*
*명지대학교 컴퓨터공학과
e-mail : cute_luna@mju.ac.kr

Study of Performance deterioration Factor for Enterprise Application based on J2EE

Min-kyu Kim*, Sung-woon Choi*
*Dept. of Computer Engineer, Myongji University

요 약

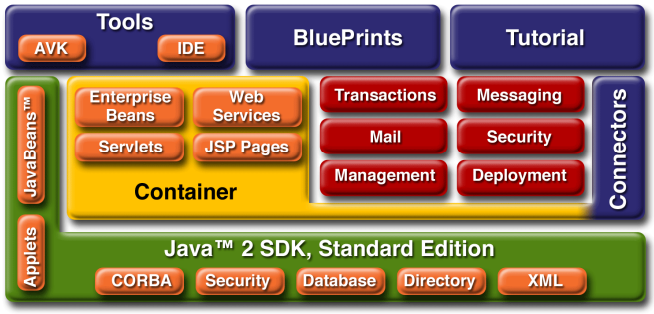
최근 엔터프라이즈 애플리케이션을 개발하는데 있어서 J2EE 이 중심 기술로 부각되었다. J2EE 의 핵심 기술이라고 할 수 있는 EJB 는 컴포넌트 기반 분산 객체 시스템을 개발하는데 있어서 생산성 향상 및 유지 보수성 등에 있어서 장점이 있지만, 성능 등에 문제점이 발견되었다. J2EE 기반으로 개발된 엔터프라이즈 애플리케이션의 성능 저하 부분을 측정하여 그 요인을 알아낸다.

1. 서론

최근 컴포넌트 기반 개발 (Component Based Development) 방법이 각광을 받고 있다. 특히 엔터프라이즈 애플리케이션을 개발하는데 J2EE 가 중심 기술로 부각되었다. J2EE 의 핵심 기술이라고 할 수 있는 Enterprise JavaBeans 의 컴포넌트 모델이 분산 객체 시스템을 개발하는데 생산성 향상 및 유지 보수성 등의 장점이 있다[9]. 하지만 성능 등의 문제점이 발견되었다[10]. EJB 를 사용하여 개발된 엔터프라이즈 애플리케이션에서 성능에 영향을 미치는 구간과 요소에 대해서 알아보고 성능 저하를 개선할 수 있는 방법을 제시한다. 먼저 2 장에서는 분산 엔터프라이즈 애플리케이션 개발을 위한 J2EE 아키텍처와 EJB 의 개요 그리고 성능을 측정하는 방법에 대해서 알아보고, 3 장에서는 실제 성능을 측정하는 도구와 기준 그리고 그 범위에 대해서 알아본다. 4 장에서는 성능 측정에 대한 결과와 성능에 미치는 요소를 알아보고 마지막 5 장에서는 결론 및 향후 연구 과제를 제시한다.

2. J2EE 아키텍처

J2EE 는 Sun 에서 개발한 기업 환경에서 서버 측 시스템을 개발할 때 사용하는 도구를 말한다. J2EE 는 java2 Standard Edition 을 비롯한 JNDI, JDBC, JTA Java Server Page, EJB 등 많은 요소를 포함하고 있으며 전체적인 아키텍처 그림은 <그림 1>과 같다.



<그림 1. J2EE 아키텍처 [3]>

J2EE 의 클라이언트 측 프로그래밍 모델은 웹 브라우저와의 상호 통신에 초점을 맞추고 있다. 이러한 클라이언트 측 프로그래밍은 크게 Java Applet, Java Servlet, Java Server Page 등의 세 가지로 나눌 수 있다. 하지만 성능 등의 문제로 Java Applet 은 실제로 많이 사용되고 있지는 않다.

J2EE 플랫폼에서 중간 계층을 구성하는 기술은 바로 J2EE 의 핵심 기술이라고 할 수 있는 엔터프라이

즈 자바빈즈(EJB)이다. EJB 가 제공하는 기능은 컴포넌트의 인스턴스를 공유함으로써 가용성을 높이고, 중간 계층 측의 보안을 강화하며, 트랜잭션 처리를 자동으로 해 주는 등 여러 가지가 있다.

J2EE 는 여러 가지 엔터프라이즈 API 를 제공하고 있는데, 관계형 데이터베이스에 접근할 때 사용되는 JDBC, 네이밍 서비스와 디렉토리 서비스에 접근할 때 사용되는 JNDI 그리고 비동기적인 워크플로우를 처리하는데 사용되는 JMS 등이 있다.

3. Enterprise JavaBeans

엔터프라이즈 자바빈즈(EJB)는 커다란 J2EE 아키텍처의 핵심 기술이며, 크게 세션 빈과 엔티티 빈, 두 가지의 빈(Bean)이 있다.

세션 빈은 비즈니스 로직을 포함하고 있는 재사용이 가능한 컴포넌트이다. 서로 다른 빈과의 상호작용이 가능하며 워크플로우를 표현하거나 특정 과제를 구현하는데 주로 쓰인다. 세션 빈은 공유된 데이터를 액세스 하는 역할을 한다. 세션 빈은 무상태 세션 빈과 상태 유지 세션 빈으로 나누어진다. 무상태 세션 빈은 관련된 서비스를 하는 메소드의 집합으로 상태 정보를 유지하지 않는 세션 빈이다. 클라이언트가 같은 메소드를 여러 번 호출한다고 해도 이전 상태 정보와 무관한 새로운 정보를 반환하게 된다. 즉 상태를 계속 유지하지 않는 대화 상태가 없는 세션 빈을 말한다. 무상태 세션 빈은 다수의 클라이언트와 대응하므로 오래 지속이 가능하며 서버의 자원에 대한 부담감을 줄일 수 있다.

상태 유지 세션 빈은 클라이언트 입장에서 클라이언트의 대리인으로써 클라이언트 애플리케이션의 연장이 되는 역할을 수행하고 있으며 상태를 계속 유지하는 세션 빈이다. 엔티티 빈과 무상태 세션 빈의 중간적인 성격을 띠고 있다. 상태 유지 세션 빈은 인스턴스가 만들어진 후 EJB 객체에 할당 되고 할당된 EJB 객체에만 서비스를 제공한다. 따라서 EJB 객체의 연결이 종료되면 빈 인스턴스도 같이 종료된다.

엔티티 빈은 데이터를 객체화 하여 재사용이 가능한 컴포넌트를 말한다. 영속성을 지닌 관계형 데이터베이스에 저장되며 빈 인스턴스와 데이터베이스의 데이터가 동기화되어야 한다. 데이터에 대한 접근을 단순화하고 일관성 있게 보장해주며 개발자로 하여금 보다 편리하고 효율적인 개발을 가능하게 해 준다. 엔티티 빈에는 컨테이너가 모든 트랜잭션을 관리하는 Container Managed Persistence(CMP)와 개발자가 직접 트랜잭션 처리를 관리하는 Bean Managed Persistance(BMP)의 두 종류로 나누어진다.

4. 성능 측정 방법과 측정 요소

분산 엔터프라이즈 애플리케이션은 웹 애플리케이션 서버 (Web Application Server, 이하 WAS) 상에서 동작한다. 따라서 WAS 의 성능이 전체 시스템의 성능에 영향을 주게 된다. 또한 개별 컴포넌트의 성능이 전체 시스템의 성능에 많은 영향을 끼치게 된다. 따라서 개별 컴포넌트의 성능을 확인하는 것은 전체 시스템의 성능을 정확히 확인하는데 있어서 매우 중요하다. 하지만 현재 개별 컴포넌트의 성능을 측정하기 위한 표준화된 방법이 존재하지 않는다. 따라서 정확한 시스템의 성능을 측정하기 위해서는 미들웨어에 독립적인 성능 측정 방법을 사용하여 개별 컴포넌트의 성능과 전체 시스템의 성능을 측정해야 한다. 본 논문에서는 Basker 등이 제안한 non-intrusive 한[1] 모델을 사용하여 배치된 컴포넌트에 영향을 주지 않는 non-intrusive 모델을 사용하여 전체 시스템의 성능을 측정할 것이다.

컴포넌트의 성능을 측정하는 요소는 크게 두 가지로 나누어서 측정한다. 첫째는 컴포넌트가 포함하고 있는 각각의 메소드별 응답시간이다. 이것은 사용자가 요구하는 컴포넌트의 기능별 성능을 나타내는 기준이 된다. 메소드별 응답시간은 메소드가 호출된 시점에서부터 메소드의 결과값이 반환된 시점까지의 시간 차를 의미한다.

$$T_{MRT} = T_{ODT} + T_{EXT}$$

$$T_{MRT} = \text{메소드 응답시간}$$

$$T_{ODT} = \text{외부요인에 의한 시간적 손실}$$

$$T_{EXT} = \text{컴포넌트 내의 메소드 로직 수행시간}$$

둘째는 컴포넌트 응답시간이다. 컴포넌트 응답시간은 컴포넌트 인스턴스 생성시간을 포함한 컴포넌트 내의 모든 비즈니스 메소드의 응답시간의 합이다. 이것은 컴포넌트 전체의 성능을 평가할 수 있다.

$$T_{CRT} = T_{CIT} + \sum_{i=0}^{n} T_{MRT} i$$

$$T_{CRT} = \text{컴포넌트 응답시간}$$

$$T_{CIT} = \text{컴포넌트의 인스턴스를 생성하는 시간}$$

$$n = \text{컴포넌트의 메소드의 개수[2]}$$

J2EE 아키텍처는 일반적으로 3 계층 형식을 취하게 되는데, 각각의 계층은 JNDI 기술을 사용하여 다른 계층의 객체를 찾아내고 RMI 를 통하여 호출하게 된다. JNDI 로 다른 계층의 객체를 찾아내는 과정에서의 걸리는 시간, EJB 의 엔티티 빈이 DBMS 의 데이터를 접근하는데 걸리는 시간 그리고 이러한 시간들이 전체 시스템에서 차지하는 비율을 알아볼 것이다.

5. 성능 측정 도구

Jakarta 프로젝트에서 개발된 jMeter[4]를 성능측정 도구로 사용할 것이다. jMeter 는 순수 자바로 개발되었으며 가상으로 부하를 생성하여 전체 시스템이 부하에 따른 전체 시스템의 성능을 측정할 수 있다. GUI 로 개발되어 전체 시스템의 성능과 그래프를 한 눈에 확인하기 좋다.

개별 컴포넌트의 성능은 java 에서 제공하는 현재 시스템 시각을 얻어오는 System.currentMills() 함수를 호출하여 각각에 대한 반응 시간을 측정할 것이다.

6. 성능 측정

6.1. EJB 계층 사이의 성능 측정

시스템 성능을 측정하는데 사용된 시스템은 인텔의 펜티엄 4 1.6Ghz, 512MB 의 메인 메모리가 사용되었으며, 마이크로소프트의 윈도 XP, Bea 의 Weblogic 8.11 그리고 데이터베이스로는 Oracle 9i 가 사용되었다. 1 차적 성능 측정은 3 계층을 구성된 시스템의 각 계층 사이의 객체를 찾아오는 부분을 측정하였다. J2EE 는 클라이언트를 JSP 와 Java 애플리케이션 두 가지로 나누어 개발할 수 있으므로 두 가지 모두 측정하였다.

<표 1. JSP 클라이언트 사용 반응시간 (단위:ms)>

Client	Session Bean
376.12	132.45

WAS 로 사용된 Weblogic 은 JSP 컨테이너를 포함한 웹 서버의 역할을 동시에 하고 있으므로 Weblogic 에서 제공하는 JSP 컨테이너를 그대로 사용하였다. <표 1>의 첫 번째 열은 JSP 로 작성된 클라이언트에서 EJB 의 세션 빈의 홈 인터페이스를 찾아오는데 걸리는 시간을 측정한 것이다. 두 번째 열은 세션 빈에서 CMP 로 작성된 엔티티 빈의 홈 인터페이스를 찾아오는데 걸리는 시간을 측정한 것이다.

<표 2. Java 애플리케이션 사용 반응시간 (단위:ms)>

Client	Session Bean
21875.56	130.22

<표 2>는 java 애플리케이션으로 클라이언트를 작성한 시스템의 반응 시간 결과이다. JSP 로 클라이언트가 만들어진 시스템에 비해서 클라이언트가 세션 빈의 홈 인터페이스를 찾아오는데 걸리는 시간이 엄청나게 오래 걸린다는 사실을 알 수 있었다. 하지만 세션 빈에서 엔티티 빈의 홈 인터페이스를 찾아오는데 걸리는 시간은 JSP 로 클라이언트가 작성된 시스템과 큰 차이를 보이지 않았다.

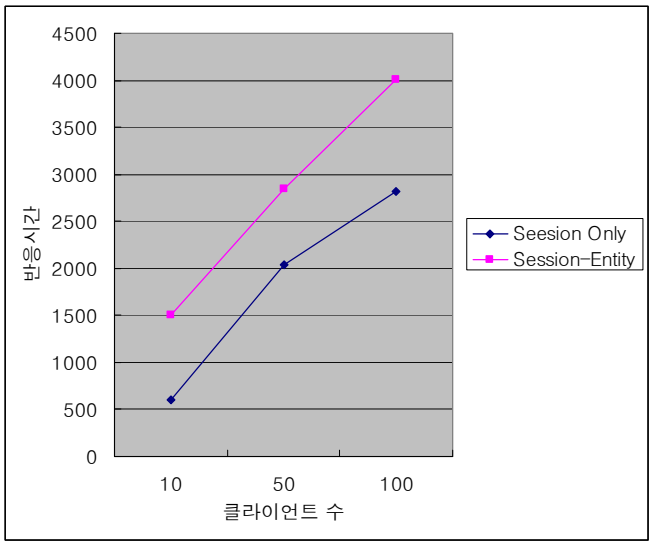
여기서 또 하나의 문제점이 발견되었는데, Java 애플리케이션으로 작성된 클라이언트는 세션 빈의 홈 인터페이스를 찾아오는 과정에서 클라이언트의 모든 기능이 정지한다는 것이다. 이는 전체 시스템에서 사용자와의 인터페이스를 제공하는 클라이언트라는 것을 생각해 봤을 때 약 20 초 간 사용자에게 대기를 강요하는 것과 다를 바 없으므로 매우 심각한 문제로 생각된다.

6.2 클라이언트 수에 따른 성능 측정

EJB 의 빈을 개발할 때는 EJBObject 인터페이스와 EJBLocalObject 라는 인터페이스 둘 중에 하나를 반드시 상속 받아야 한다. 인터페이스 이름에서도 알 수 있듯이 로컬 환경에서 사용하는 EJBLocalObject 와 EJBObject 인터페이스 사이에 성능 차이가 있는지 역시 측정하였다. 하지만 세션 빈에서는 EJBLocalObject 를 상속하여도 클라이언트가 원격에서 요청을 하게 되므로 성능에 영향을 주지 않는다. 따라서 일반적으로 WAS 와 로컬로 연결되어 있는 데이터베이스와의 통신을 담당하는 엔티티 빈을 작성하는 경우에만 EJBObejct 와 EJBLocalObject 를 따로 적용하였다.

<표 5. 클라이언트 수에 따른 반응 시간(단위:ms)>

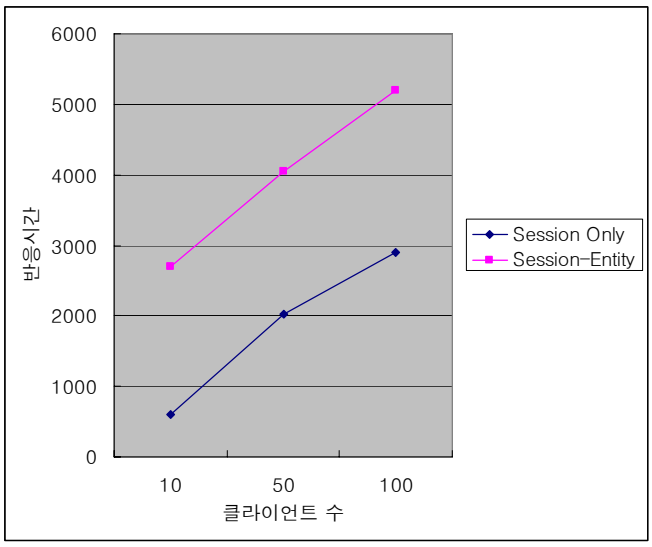
클라이언트 수	Session Only	Session-Entity
10	601.98	1502.87
50	2031.65	2844.12
100	2819.77	4001.09



사용자가 늘어남에 따라서 세션 빈만을 사용하였을 때는 처음에는 급격한 반응 시간 증가를 보이다가 50 명 이상의 사용자에서는 비교적 완만한 증가를 보이고 있다. 세션과 엔티티 빈을 모두 사용하였을 때는 역시 같은 현상이 발생하기는 하나 그 증가폭이 세션만 사용하였을 때에 비해서 적은 것을 알 수 있다. <표 5>의 측정 결과는 EJBLocalObject 를 상속받은 엔티티 빈을 구성한 결과이다. 다음 <표 6>은 EJBObject 를 상속받아 구성한 엔티티 빈의 결과이다.

<표 5. 클라이언트 수에 따른 반응 시간(단위:ms)>

클라이언트 수	Session Only	Session-Entity
10	600.88	2702.99
50	2020.30	4044.02
100	2900.01	5201.74



일단 세션 빈에서는 변동 사항이 없으므로 큰 차이를 보이고 있지 않다. 그리고 클라이언트의 수에 따라 증가하는 반응 속도의 폭도 모두 크게 변함이 없다. 하지만 EJBLocalObject 를 상속받아 구현한 엔티티 빈과 EJBObject 를 상속받아 구현한 엔티티 빈에서는 크게 1 초 이상의 성능 저하를 발견하였다.

일반적으로 엔터프라이즈 애플리케이션을 개발할 때, EJB 컨테이너 역할을 하는 WAS 와 데이터베이스 사이에 로컬로 연결이 되어 있는 경우가 많으므로, EJBLocalObject 인터페이스를 상속하여 엔티티 빈을 구성하는 것이 성능 향상에 도움이 되는 것을 알 수 있다[5].

7. 결론 및 향후 연구 과제

엔터프라이즈 애플리케이션을 EJB 로 구성할 때, 전체 성능에서 가장 큰 영향을 주는 것은 각 계층 사이에서 EJB 객체를 찾아오는 과정이라는 것을 알 수 있다. EJBLocalObject 인터페이스와 EJBObject 인터페이스 와의 성능 차이도 역시 세션 빈에서의 엔티티 빈 인스턴스를 찾아오는 과정의 문제이다. EJBObject 는 빈 인스턴스를 찾을 때, 원격에 있는 것을 가정하여 빈 인스턴스를 찾게 되므로 비교적 오랜 시간을 소비하게 되는 것이고, EJBLocalObject 인터페이스는 빈 인스턴스를 찾을 때, 로컬 영역에서만 빈 인스턴스를 찾게 되므로 성능 향상에 도움이 있었다. 또한, EJB 객체를 찾아올 때는 InitialContext 객체의 lookup 메소드를 호출하여 각 계층의 인스턴스를 찾아오게 된다. 만약 lookup 메소드를 멀티쓰레딩 하여 처리하게 된다면 성능 향상에 도움을 줄 수 있을 것으로 기대된다.

EJB 기반의 엔터프라이즈 애플리케이션 성능에 영향을 주는 것은 EJB 인스턴스를 찾아오는 부분 뿐 아니라 WAS 가 풀링하고 있는 빈 인스턴스의 개수, 쓰레드의 개수 그리고 가비지 컬렉터가 작동하는 부분 등 여러 가지 변수가 성능 저하에 영향을 줄 것이다. 따라서 향후에 이러한 요소를 모두 포함한 성능 측정 평가와 이에 따른 성능 향상 기법이 연구되어야 할 것이다.

참고문헌

[1] B.Sridharan, B. Dasarathy and A. P. Mathur, “On Building Non-intrusive Performance Instrumention Blocks for CORBA-based Distrubted Systems”, 4th IEEE International Computer Performance and Dependability Sysmposisum, Chicago March 2000.

[2] 황길승, 이긍혜, “미들웨어 독립적인 분산 컴포넌트 성능측정 도구 설계”, 정보과학회논문지: 소프트웨어 및 응용 제 31 권 제 8 호, 2004. 8

[3] http://java.sun.com/j2ee/setstandard.html

[4] http://jakarta.apache.com/jmeter

[5] 김정덕, 최성운, “성능 중심의 J2EE 아키텍처 패턴 연구”, 정보처리학회, 2004 년 춘계학술대회

[6] Ed Roman, Scott W.Ambler, Tyler Jewell, Mastering Enterprise Application JavaBeans Second Edition, Wiley

[7] Jack Shirizi, Java Performance Tuning, O’REILLY

[8] Rod Johnson, expert one-to-one J2EE Design and Development, Wrox

[9] 최시원, 김수동, “EJB 환경에서 객체지향 상속 관계 설계 패턴”, 정보처리학회논문지 D, 제 11-D 권 제 1 호 (2004, 2)

[10] 나학청. 김수동, “EJB 어플리케이션의 성능 모니터링 기법”, 정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 6 호(2003. 6)