동시접속 사용자 접근을 고려한 데이터베이스 커넥션 풀 아키텍처

Database Connection Pool Architecture for User Interconnections Access

김영찬*, 김태간*, 이세훈**, 임기욱***, 이정현****
인하대학교 정보공학과*, 인하공업전문대 컴퓨터시스템과**,
선문대학교 컴퓨터정보학부***, 인하대학교 컴퓨터공학부****

Young-Chan Kim(chanible@naver.com)*, Tae-Gan Kim(taegan@empal.com)*,
Se-Hoon Lee(seihoon@inhatc.ac.kr)**, Kee-Wook Rim(rim@sunmoon.ac.kr)***,
Jung-Hyun Lee(jhlee@inha.ac.kr)****

요약

어플리케이션의 데이터베이스 시스템 사용이 날로 증가하면서, 사용자 급증에 따른 데이터베이스 시스템에 동시접속 하는 커넥션 처리 방법에 있어 중요성이 부각되고 있다. 이러한 데이터베이스의 커넥션을 효과적으로 처리하기 위해 데이터베이스 커넥션 풀이 도입되었으나, 동시접속 사용자가 증가하는 형태에 따른 커넥션 처리는 미비한 실정이다. 본 논문에서는 기존의 커넥션 풀 아키텍처를 개선하기 위하여 주 커넥션 풀 이외에 보조 커넥션 풀을 추가하였다. 또한 두 커넥션 풀의 연결을 관리하기 위하여 커넥션 풀 브로커를 정의하였다. 커넥션 풀 브로커는 주 커넥션 풀이 한계에 도달하였을 때 보조커넥션 풀로 연결을 제어한다. 실험을 통하여 동시접속 사용자가 급격히 증가하는 형태의 요청처리에 있어, 기존 데이터베이스 커넥션 풀 구조와 비교해서 응답시간의 향상된 결과를 확인하였다.

■ 중심어 : | 데이터베이스 커넥션 풀 | 커넥션 브로커 | 커넥션 아키텍처 | JDBC |

Abstract

The use of database system in application increases day by day. This brought out the DBMS interconnection access problem with rapid increase of the user. To solve these database connection problem, database connection pool has been presented. However, there is much to be desired on user's interconnection access. To improve previous connection pool architecture, we have designed the sub connection pool besides main connection pool in this paper. We defined connection pool broker to manage both main and sub connection pool connection. When main connection pool reached its limitation, connection pool broker transfers a connection from a main connection pool to a sub connection pool. When the interconnection access user increased rapidly, we have proved that the suggested sub pool architecture is more effective on response time by comparing it with other existing DBMS connection pool architectures using simulation.

■ keyword : | Database Connection Pool | Connection Broker | Connection Architecture | JDBC |

* 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었습니다.
(IITA-2008-C1090-0801-0020)
접수번호 : #081008-006
접수일자 : 2008년 10월 08일
심사완료일 : 2008년 11월 04일
교신저자 : 김영찬, e-mail : chanible@naver.com

I. 서 론

IT 산업이 발전하고 인터넷이 보급화 됨에 따라 웹 어플리케이션들은 많은 데이터를 취급하게 되었으며, 또한 동적인 처리를 위하여 데이터베이스 자원을 참조하지 않는 것은 거의 없을 것이다.

최근 들어 가장 많이 활용되는 JSP/SERVLET, EJB와 같은 JAVA기반의 웹 어플리케이션들은 데이터베이스를 참조하기 위하여 JDBC(Java Database Connectivity)를 사용하게 된다[1]. JDBC 드라이버는 데이터베이스에 커넥션을 맺고, 웹 어플리케이션의 질의 요청에 따른 결과 값을 제공하여 준다.

웹 어플리케이션은 다양한 형태로의 업무환경을 구성하게 되는데 대부분이 데이터베이스 자원을 사용하며, 이 데이터베이스 자원을 사용하는 것이 가장 많은 시간을 필요로 하는 부분이다. 즉, 데이터베이스에 연결하여 세션을 얻는 시간이 가장 오래 걸리고 시스템 리소스를 많이 소모하는 부분이다. 웹 어플리케이션의 특성상 연결을 얻고 해제하는 작업은 클라이언트들 마다 각각 빈번하게 발생하게 되므로 서버의 입장에서는 무수히 많은 데이터베이스 연결 및 해제 작업을 하게 되므로 오버헤드가 많이 발생하게 된다[2].

데이터베이스 커넥션 풀이란 이렇게 클라이언트가 데이터베이스에 접근할 때마다 연결하여 세션을 생성하고 작업 후에 다시 연결을 해제하는 과정에서의 오버헤드를 줄이기 위하여 데이터베이스의 접근 속도를 높여보자는 생각에서 시작된 개념이다[3]. 한번 생성된 연결은 작업을 마치고 바로 해제하는 것이 아니라 연결을 풀이라는 공간에 보관해 두고 이후 다시 클라이언트의 연결이 요청되면 이를 재사용 하는 방법이다[4]. 그러나 이와 같은 구조에서도 데이터베이스 커넥션 풀에 대한 문제가 다음과 같이 존재하고 있다.

데이터베이스 커넥션 풀의 커넥션 사용 증가율은 웹 어플리케이션의 동시접속 사용자에 비례하며, 이에 따라 웹 어플리케이션 서버의 리소스가 동시에 폭발적으로 증가하게 되는데 이는 EJB와 JSP, SERVLET 등과 같은 웹 어플리케이션들에게도 한정된 자원에서의 리소스 사용으로, 결국 웹 어플리케이션 서버 전체는 리소스 고갈로 인하여 클라이언트들의 요청을 처리함에 있어 지연현상이 발생하거나, 처리하지 못하는 장애현상이 발생 하게 된다.

실제로 인터넷 기반의 원서접수시스템이나 예약시스템등과 같은 사용자의 접속이 일시에 폭발적으로 증가하는 형태의 인터넷 서비스는 동시접속현상이 발생하여 증가함에 따라 서버가 다운이 되는 현상이 빈번히 발생하였다[10-12]. 이처럼 데이터베이스를 연동하는 웹 어플리케이션에서 데이터베이스 커넥션의 관리방법은 전체 시스템의 안전성 및 성능에 밀접한 관련이 있기 때문에 이를 개선하는 일이 중요하다. 현재의 데이터베이스 커넥션 풀 아키텍처는 웹 어플리케이션 서버 내에서 관리하고 있고 이는 제한된 자원에서 어플리케이션들과 데이터베이스 커넥션 풀이 서로 경쟁관계를 이루고 있기 때문에 이에 대한 개선이 필요하다[1].

본 논문에서는 웹 어플리케이션 구축 시 사용하는 데이터베이스 커넥션 풀에 대하여 사용자의 동시접속에 따른 데이터베이스 접근에 대한 성능을 평가하고, 이에 따른 개선된 데이터베이스 커넥션 풀 아키텍처를 제시하고 구현 하는 것을 목표로 한다. 구현된 데이터베이스 커넥션 풀 아키텍처는 사용자의 동시접속 부하를 발생하게 하여 그 성능을 측정함으로써, 구현된 데이터베이스 커넥션 풀이 안정적으로 요청을 처리 할 수 있는가를 파악한다.

II. 관련연구

본 장에서는 JDBC의 전반적인 개요와 JDBC아키텍처에 대하여 설명하고, 이를 활용한 데이터베이스 커넥션 풀의 연결 아키텍처에 대하여 설명한다.

1. JDBC

일반적으로 JDBC라 함은, 테이블로 이루어진 데이터를 사용할 수 있도록 하는 Java API를 말한다[5]. JDBC는 자바 어플리케이션의 데이터베이스 시스템에 대한 이식성을 높이기 위하여 설계된 것이다. 즉, JDBC API를 사용한 어플리케이션은 특정 데이터베이스관리시스

템에 종속되지 않으며, 수정 없이 Sybase, Oracle, SQL Server등 다양한 데이터베이스관리시스템을 이용할 수 있도록 한다[5]. 자바기반의 어플리케이션들은 데이터베이스에 접근하기 위하여 JDBC API를 사용하며, 이는 각각의 벤더 업체들에 의해서 구현·제공하고 있다.

JDBC API는 JDBC 핵심 API와 JDBC 확장 API로 분류된다. JDBC 핵심 API는 어플리케이션이 데이터베이스에 성공적으로 접근하기 위해 반드시 필요한 기능을 지원하는 데 초점을 맞춘 반면, JDBC 확장 API는 데이터베이스 접근을 위한 필수 기능은 아니나, 성능과 프로그램 작성의 편리를 위해 JNDI API를 이용한 네이밍 서비스, JDBC드라이버 기반 연결 풀, 분산 트랜잭션, 로우셋 등의 기능을 지원한다[3].

데이터베이스관리시스템 벤더들은 JDBC API에 명세된 대로 자신들의 데이터베이스와 통신할 수 있는 소프트웨어인 JDBC드라이버를 만들어서 배포한다. 이 드라이버는 자바로 만들어진 클래스 파일의 묶음으로 jar나 zip형태로 패키지 되어있다. 이러한 JDBC 드라이버는 JDBC에 정의된 자바명령어를 그들만의 데이터베이스에 맞는 명령어로 바꾸어 전달하고 그 결과로 넘겨받은 값을 다시 JDBC 드라이버를 통하여 자바 코드에 전달하는 어뎁터(adapter)이다[9].

[그림 1]은 JDBC 아키텍처를 설명한 그림이다[9]. 자바 어플리케이션의 개발은 JDBC 드라이버를 제공하는 모든 데이터베이스를 단일한 인터페이스(JDBC API)로써 접근이 가능하기 때문에 CRUD가 데이터베이스시스템에 종속적이지 않는 장점을 가지고 있다.

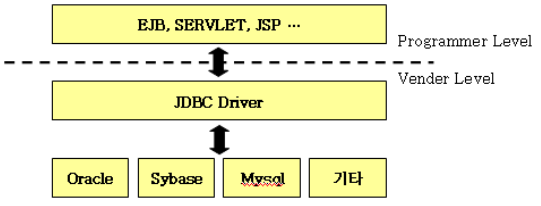

그림 1. JDBC 아키텍처

JDBC API를 이용하여 데이터베이스를 접근하는 환경은 2-tier와 3-tier 모델로 분류할 수 있다[1].

- Two-tier 아키텍처

Two-tier 아키텍처에서는 JSP/SERVLET, EJB와 같은 자바 기반의 어플리케이션들이 데이터베이스에 직접 연결된다. 여기서는 해당 데이터베이스관리시스템과 통신 할 수 있는 JDBC드라이버를 필요로 한다. JDBC 드라이버를 이용하여 클라이언트의 SQL문을 데이터베이스로 전달하고, 처리 결과는 다시 클라이언트로 돌아온다. 데이터베이스와 클라이언트는 네트워크를 통해 연결되어 있는 서로 다른 컴퓨터에 위치할 수도 있다.

- Three-tier 아키텍처

Three-tier 아키텍처의 경우, Two-tier 아키텍처에 Middle-tier를 추가한 것으로 SQL문은 Middle-tier로 보내고, 이는 다시 SQL 문장을 데이터베이스관리시스템에 보낸다. 데이터베이스관리시스템은 SQL문장을 처리하여 결과를 Middle-tier에 보내고, 이는 다시 클라이언트에 전달된다. Middle-tier는 엔터프라이즈 환경을 위해 sun에서 제안한 J2EE 스펙을 구현한 어플리케이션 서버를 이용한다[1].

2. 데이터베이스 커넥션 풀

웹 응용과 같이 다수의 클라이언트가 빈번히 데이터베이스에 접근을 요구하는 환경에서, 데이터베이스 접근을 필요로 하는 모든 클라이언트는 각각 데이터베이스 연결을 생성하고, 이 연결을 이용하여 데이터베이스 작업을 수행한 후 연결을 해제하는 것은 데이터베이스관리시스템 입장에서 매우 큰 오버헤드가 되어 성능을 저하시키는 원인이 된다[5].

데이터베이스 커넥션 풀이란 이러한 문제의 해결책으로 최소한의 커넥션만 미리 만들어 놓고, 클라이언트의 요청을 처리하고 사용이 끝나면 그 커넥션을 다른 클라이언트들이 재사용이 가능하도록 하는 구조를 말한다. 기본적인 데이터베이스 커넥션 풀에 의한 연결 아키텍처는 다음 [그림 2]와 같다[4].

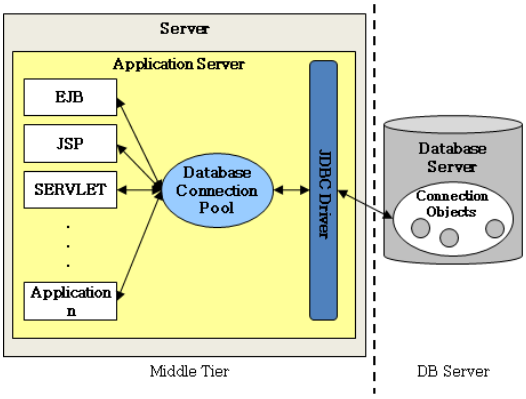

그림 2. 데이터베이스 커넥션 풀 연결 아키텍처

이러한 구조는 동시에 여러 클라이언트의 요청을 데이터베이스 커넥션 풀에 대기 중인 커넥션으로부터 할당 받고 대기 중인 커넥션이 데이터베이스 커넥션 풀에 없을 경우 추가로 커넥션을 데이터베이스 커넥션 풀에 할당하여 이를 사용하게 된다. 그리고 사용이 끝난 커넥션은 데이터베이스 커넥션 풀에 회수 하여 재사용을 하게 된다. 일반적인 데이터베이스 커넥션 풀의 관리 과정은 다음과 같다[7].

가. 데이터베이스 커넥션 풀에서 커넥션 얻기
  ① 최초 요청시 일정수의 커넥션을 생성해서 데이터베이스 커넥션 풀에 보관
  ② 클라이언트(어플리케이션)가 커넥션을 요구
  ③ 커넥션 제공 및 사용
나. 데이터베이스 커넥션 풀에서 커넥션 반환
  ① 커넥션 반환
  ② 반환된 커넥션을 커넥션 풀에 저장
다. 데이터베이스 커넥션 풀에 의한 이점
  ① 커넥션을 미리 생성해 놓고 요청시 바로 데이터베이스 커넥션 풀로부터 사용함으로써 커넥션 생성시간을 절약할 수 있다.
  ② 최소한의 커넥션을 생성하고 사용함으로써 자원을 효율적으로 사용할 수 있다.
  ③ 생성된 커넥션은 사용 후에 데이터베이스 커넥션 풀에 보관되어 재사용 된다.

이러한 이점에도 불구하고 근본적으로 구조적인 문제점이 있음을 알 수 있다. 데이터베이스 커넥션 풀은 어플리케이션 서버의 자원으로서 종속적이며, 어플리케이션이 할당하고 있는 자원에 한하여 커넥션을 생성할 수 있다는 것을 알 수 있다. 또한, 데이터베이스 커넥션 풀의 생성 개수에 있어서 최대 생성할 수 있는 커넥션이 명시되어야 하는데, 이 최대 커넥션의 개수가 사용량보다 초과 하였을 경우 문제가 발생할 수 있다.

ASP 서비스와 같은 커넥션이 빈번히 발생하거나 폭발적으로 증가하는 형태의 서비스는 한정된 자원을 효율적으로 활용하여 최대한 많은 커넥션을 데이터베이스 커넥션 풀에 확보하여야 한다.

III. 데이터베이스 커넥션 풀 아키텍처 설계 및 구현

본 장에서는 기존의 데이터베이스 커넥션 풀의 제약된 문제점을 개선하기 위하여, 주 커넥션 풀 이외에 보조 커넥션 풀과 이를 관리하기 위한 커넥션 풀 브로커를 정의하여, 개선된 데이터베이스 커넥션 풀 아키텍처를 설계하고 이를 구현한다.

1. 데이터베이스 커넥션 풀 아키텍처 설계

[그림 3] 은 본 논문에서 설계한 개선된 데이터베이스 커넥션 풀 연결 아키텍처이다.

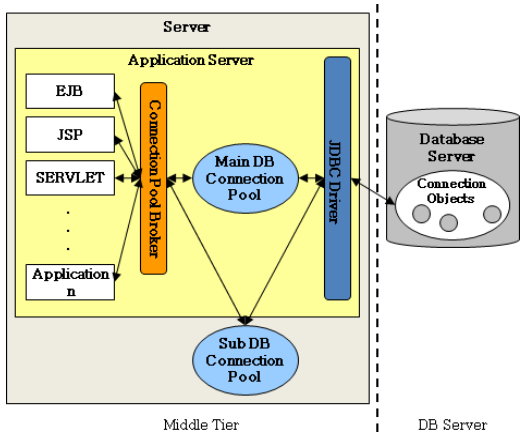

그림 3. 개선된 데이터베이스 커넥션 풀 연결 아키텍처

EJB나 JSP/SERVLET등과 같은 웹 어플리케이션들은 커넥션 브로커에게 커넥션할당을 요청하고 커넥션 브로커는 어플리케이션 서버 내의 메인 데이터베이스 커넥션 풀로부터 커넥션을 얻어오거나, 서버의 보조 데이터베이스 커넥션 풀로부터 커넥션을 얻어오는 구조로 되어 있다. 그리고 이 데이터베이스 커넥션 풀들은 JDBC 드라이버를 통하여 해당 데이터베이스관리시스템에게 커넥션을 요청하여 할당 받는다.

보조 데이터베이스 커넥션 풀은 어플리케이션 서버에 종속적이지 않고 독립적으로 존재하며, 기존의 메인 커넥션 풀의 증가에 따른 부하 발생 시 보조 데이터베이스 커넥션 풀로부터 커넥션을 할당받을 수 있도록 전체적인 데이터베이스 커넥션 풀 아키텍처를 개선하였다.

1. 데이터베이스 커넥션 풀 아키텍처 구현

[그림 4] 는 본 논문에서 제시한 커넥션 풀 브로커의 아키텍처이다. 동시접속 사용자 접근을 고려한 데이터베이스 커넥션 풀 구현에 있어 커넥션 풀 브로커를 정의하여, 메인 커넥션 풀과 별도로 보조 커넥션 풀을 두어 부하를 분산하게 된다.

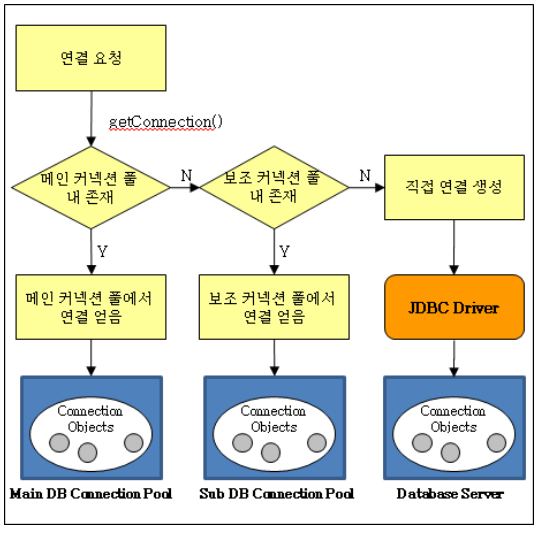

그림 4. 커넥션 풀 브로커 아키텍처

[표 1] 은 커넥션 풀 브로커에 대한 핵심 처리과정을 나타내고 있다. 웹 어플리케이션의 커넥션 요청을 처리하기 위하여 getConnection()에 의한 커넥션 요청은 커넥션 풀 브로커에게 전달되어진다. 커넥션 풀 브로커는 메인 커넥션 풀에 커넥션이 존재하는지 확인하는 과정을 거치며, 메인 커넥션 풀에 대기 중인 커넥션이 존재하면, 메인 커넥션 풀로 부터 이를 할당하게 된다. 만약 메인 커넥션 풀에 대기 중인 커넥션이 존재하지 않는다면 보조 커넥션 풀로부터 대기 중인 커넥션을 확인하게 되는데, 보조 커넥션 풀에 대기 중인 커넥션이 존재하면 보조 커넥션 풀로부터 커넥션을 할당받고, 만약 보조 커넥션 풀에서도 미리 생성된 커넥션이 존재 하지 않는다면 JDBC Driver를 통하여 직접 데이터베이스 서버로부터 커넥션을 생성하여 이를 할당하게 된다.

표 1. 커넥션 풀 브로커 핵심 처리과정

```
INITIAL Connection Object FROM getConnection Function

FUNCTION getConnection(Connection)
    INITIAL MainPool Object FROM DBCP
    INITIAL SubPool Object FROM DBCP
    IF (Connection Exist FROM MainPool)
        THEN
            GET Connection FROM MainPool
        ELSE
            IF (Connection Exist IN SubPool)
                THEN
                    GET Connection FROM SubPool
                ELSE
                    GET Connection FROM JDBC Driver
            END IF
    END IF
    RETURN Connection
END FUNCTION
```

Ⅳ. 실험 및 평가

본 장에서는 실험방법을 제시하고, 관련연구에서 설명한 데이터베이스 커넥션 풀을 대상으로 그 성능을 평

가하고, 3장에서 구현한 동시접속 사용자 접근을 고려한 데이터베이스 커넥션 풀과의 차이점에 대하여 기술한다.

1. 실험 환경

개선된 데이터베이스 커넥션 풀 아키텍처의 구현에 사용된 서버는 HP DL380 기종을 사용하였다. 개발 툴로는 eclipse3.3을 사용하였고, 개발 언어로는 HTML, Java Script, Java, Servlet을 사용하였으며, DBCP와 JNDI 기술을 활용하여 커넥션 풀을 구현하였다. 데이터베이스 접속환경을 마련하기 위해서 oracle 10g 데이터베이스관리시스템과 oracle10g JDBC 드라이버를 사용하였고 HTML과 Java 코드를 분리하기 위해 MVC 기법을 사용하였다. 이 시스템의 구현환경을 살펴보면 다음과 같다.

- System : Intel Xeon 2GHz * 8, RAM 4GB
- OS : openSuSE Linux 10.3
- DBMS : Oracle 10g
- JAVA : J2sdk1.6
- Web Server : Apache2
- Servlet Engine : Tomcat 6.0

보통 웹 어플리케이션 서버의 구성이 Three-tier인 점을 감안해 Three-tier구조로 구현을 하였다. 웹서버는 브라우저로부터 클라이언트의 요청을 받아 서블릿 엔진에 전달하고, 서블릿 엔진은 다시 데이터베이스관리시스템에게 요청을 전달하고 이를 다시 응답 받는 형태로 구현하였다.

2. 실험 방법

본 논문에서는 데이터베이스 커넥션을 풀로부터 얻어 처리하는 웹 어플리케이션의 처리 시간을 얻기 위하여 동시 접속자를 늘려가면서 그 성능을 측정 하였다. 측정의 대상은 관련연구에서 설명한 데이터베이스 커넥션 풀 아키텍처와 본 논문에서 제시한 데이터베이스 커넥션 풀 아키텍처이다. 성능 측정 도구로는 Apache JMeter를 사용 하였다. Apache JMeter는 테스트 기능과 퍼포먼스를 측정하는 기능을 갖는 100% 순수 자바 데스크탑 어플리케이션이다. 측정도구를 사용함으로써 사용자의 행위나 데이터의 크기, 유형에 영향을 받지 않는 순수한 조건에서의 결과를 도출한다.

주 커넥션 풀의 크기는 Default로 설정 되어 있는 20으로 설정하였고, 보조 커넥션 풀의 크기도 주 커넥션 풀의 크기와 동일하게 20으로 설정 하였다. 동시접속자의 수는 10, 20, 50, 100, 200 으로 설정하여 각각의 평균 응답 시간을 측정하였다. 클라이언트의 평균 응답시간은 다음과 같다.

$$\frac{\sum_{dient=0}^{n} Endtime_{dient} - Starttime_{dient}}{n} \quad (1)$$

여기서 n 은 클라이언트의 수로 평균 응답시간은 클라이언트의 요청처리시간의 합을 n으로 나눈 것이다.

3. 실험 평가

데이터베이스 커넥션 풀의 실험 결과는 다음과 같다. [그림 5] 는 평균 응답시간을 측정한 결과를 그래프로 나타낸 것이다.

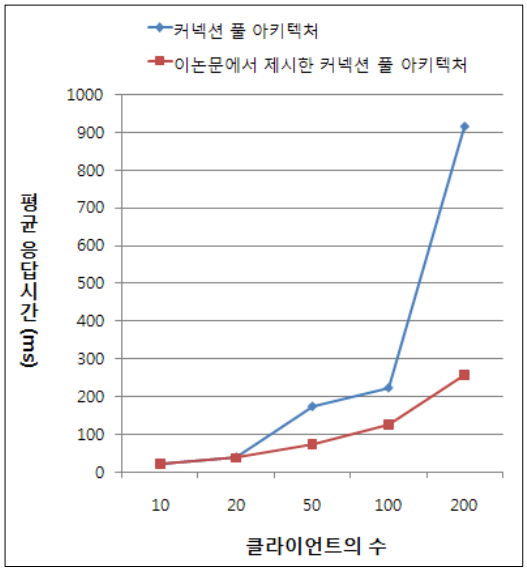

그림 5. 평균 응답시간

평균 응답시간 결과그래프를 보면 클라이언트의 동시접속자 수가 증가할수록 데이터베이스 커넥션 풀 아키텍처 방식과, 본 논문에서 제시한 데이터베이스 커넥션 풀 아키텍처 방식 모두 응답시간은 점차적으로 증가하는 것을 알 수 있다. 동시접속자의 수가 20까지 증가할 경우는 두 커넥션 풀 아키텍처의 방식은 특별한 차이점이 없으나, 그 이상의 동시접속자의 수가 발생할수록 기존의 데이터베이스 커넥션 풀 아키텍처는 응답시간이 급격히 증가하여 900ms를 넘었고, 본 논문에서 제시한 커넥션 풀 아키텍처는 250ms로 동시접속자의 수가 200일 경우 그 차이는 약 380% 가량 성능 향상을 보였다.

[그림 6] 은 최소 응답시간을 측정한 결과를 그래프로 나타낸 것이다.

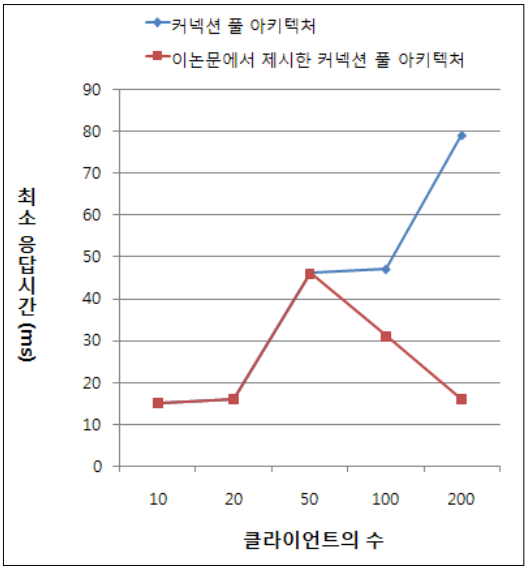

그림 6. 최소 응답시간

최소 응답시간 결과그래프를 보면 동시접속자의 수가 50 이상이 되면서부터 기존의 커넥션 풀 아키텍처는 최소 응답시간이 계속적으로 증가하는 형태로 80ms 까지 올라가는 그래프를 볼 수 있다. 본 논문에서 제시한 커넥션 풀 아키텍처는 실험에서 최대 200의 동시접속자 속에서도 최소 응답시간은 특별히 증가하지 않았고, 최대 48ms 내에서 최소 응답시간을 유지하고 있다.

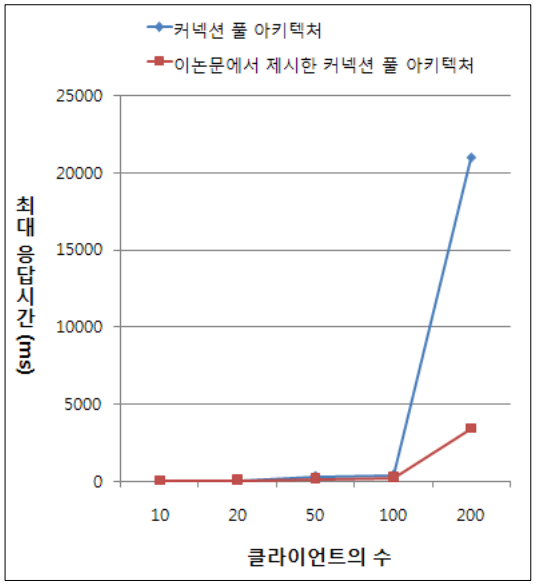

그림 7. 최대 응답시간

[그림 7] 은 최대 응답시간을 측정한 결과를 그래프로 나타낸 것이다. 최대 응답시간 결과그래프를 보면 동시접속자의 수가 100 이 되는 시점까지는 두 커넥션 풀 아키텍처 방식은 특별한 차이점이 없으나, 동시접속자의 수가 100 이상이 되면서부터 기존의 커넥션 풀 아키텍처는 최대 응답시간이 계속적으로 증가하는 형태로 최대 21200ms를 기록하였고, 본 논문에서 제시한 커넥션 풀 아키텍처는 최대 200의 동시접속자 속에서 최대 응답시간이 3800ms로 기존의 커넥션 풀 아키텍처보다 약 500% 가량 향상된 성능을 보이고 있다.

마지막으로 [그림 8] 은 에러 발생비율을 측정한 결과를 그래프로 나타낸 것이다.

기존의 커넥션 풀 아키텍처에서 동시접속자의 수가 200일 경우 에러가 1건이 발생 하였고, 본 논문에서 제시한 커넥션 풀 아키텍처에서는 에러발생이 0건으로 동시접속자에 대한 응답처리가 안정적으로 이루어졌다.

이처럼, 보조커넥션 풀을 추가함으로써 동시접속 사용자의 요청에 대한 응답시간은 크게 향상되었으며, 안정적인 처리 또한 가능하다는 것을 확인하였다.

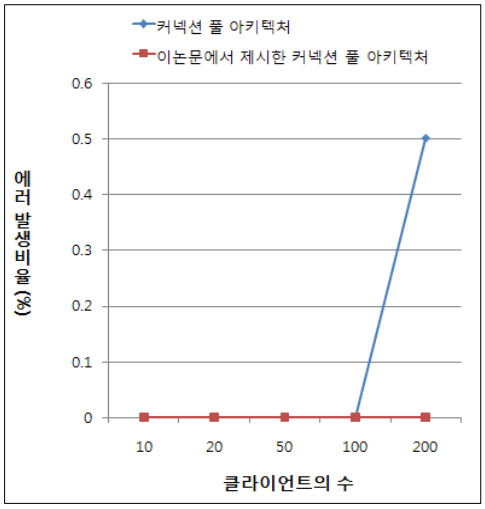

그림 8. 에러 발생비율

V. 결론

본 논문에서는 동시접속 사용자 접근을 고려한 데이터베이스 커넥션 풀 아키텍처를 제시하고 이를 구현하였다. 실험 결과를 통하여 기존의 데이터베이스 커넥션 풀 아키텍처 방식은 동시접속 사용자가 급속히 증가함에 따라 데이터 처리에 대한 응답시간 또한 급속히 증가하는 것을 알 수 있다. 반면, 본 논문에서 제시한 커넥션 풀 아키텍처 방식은 동시접속 사용자가 증가함에 따른 데이터 처리 응답시간이 기존의 데이터베이스 커넥션 풀 방식에 비하여, 평균 응답시간은 약 300% 가량 향상된 결과를 보여주었고, 최소 응답시간은 약 100% 가량 향상된 결과를, 최대 응답시간은 약 500% 가량 향상된 결과를 보여주었다. 에러발생비율에 있어서도 본 논문에서 제시한 커넥션 풀 아키텍처에서는 0%로 기존의 커넥션 풀 아키텍처보다 안정적으로 동시접속 사용자에 대한 데이터 응답 처리가 이루어진다는 것을 실험을 통하여 확인하였다. 이는 데이터베이스 서버에서 연결을 얻어 세션을 생성하고 해제하는 작업에 대하여 많은 리소스를 소요하는 것을 주 커넥션 풀과 보조 커넥션 풀에 의해 보관되어진 커넥션 리소스를 재사용하여 연결과 관련된 오버헤드를 줄였고, 동시접속 사용자가 증가함에 따라 주 커넥션 풀에서 처리용량의 한계점에 올랐을 때, 보조 커넥션 풀이 그 역할을 보조함으로 전체적인 시스템 성능에 있어 처리 용량의 한계를 끌어올려 데이터 요청의 응답시간에 있어서 좋은 결과를 나타내었다.

본 논문에서 제시한 데이터베이스 커넥션 풀 아키텍처는 어플리케이션 서버에서 커넥션 풀에 대기중인 커넥션의 수가 많을수록 동시접속 사용자에 대한 데이터 처리 응답시간이 빨라지고 안정적으로 서비스를 할 수 있다는 것을 실험을 통하여 확인하였다.

향후 연구과제로는 동적 로드밸런싱 기법을 적용하여 하나의 데이터베이스 커넥션 풀에 부하가 집중되는 것을 분산하여야 할 것이다.

참 고 문 헌

[1] Java.sun.com, *JDBC Architecture*, Sun Microsystems, 2008.

[2] Marc A. Mnich, *Java Servlet Technologies*, JavaExchange.com, 2002.

[3] E. Jon and H. Linda, *JDBC 3.0 Specification Final Release*, Sun Microsystems, 2001.

[4] Java.sun.com, *Dive into connection pooling with J2EE*, Sun Microsystems, 2000.

[5] R. George, *Database Programming with JDBC and Java, J2EE*, O'Reilly, 2000.

[6] W. Seth and H. Mark, *JDBC 2.1 API*, Sun Microsystems, 1999.

[7] 최영관, 김대성, 김완기, 한창헌, *소설 같은 JSP*, JABOOK, 2002.

[8] 천기숙, *JDBC 연결 풀 관리자 설계 및 구현*, 성신여자대학교 대학원, 2002.

[9] 김영찬, 이세훈, "동시사용자 접근이 용이한 데이터베이스 커넥션 풀 아키텍처", 한국컴퓨터정보학회 하계학술발표논문집, 제16권, 제1호, pp.125-130, 2008.

[10] http://article.joins.com/article/article.asp?total_id=2889936

[11] http://www.newsen.com/news_view.php?uid=200810241124301001

[12] http://www.ytn.co.kr/_ln/0103_200809271433029147

저 자 소 개

김 영 찬(Young-Chan Kim) 정회원

- 2007년 8월 : 평생교육진흥원 컴퓨터공학과(공학사)
- 2007년 9월 ~ 현재 : 인하대학교 공학대학원 정보공학과(석사과정)
- 2006년 9월 ~ 현재 : (주)케이티하이텔솔루션 부설연구소 연구원

<관심분야> : 데이터베이스, 소프트웨어아키텍처, 분산시스템

김 태 간(Tae-Gan Kim) 정회원

- 2003년 : 서울산업대학교 컴퓨터공학과(공학사)
- 2005년 : 인하대학교 교육대학원 정보컴퓨터교육(교육학석사)
- 2005년 : 인하대학교 대학원 정보공학과 박사과정

<관심분야> : 임베디드시스템, 센서네트워크, 소프트웨어공학

이 세 훈(Se-Hoon Lee) 정회원

- 1985년 : 인하대학교 전자계산학과(이학사)
- 1987년 : 인하대학교 대학원 전자계산학과(이학석사)
- 1996년 : 인하대학교 대학원 전자계산공학과(공학박사)
- 1987년 ~ 1990년 : 해병대 분석장교
- 1990년 ~ 1993년 : (주)비트컴퓨터연구소 연구원
- 1999년 : 멀티미디어기술사
- 2001년 ~ 2002년 : 미국 뉴저지공과대학(NJIT) 교환교수
- 1993년 ~ 현재 : 인하공업전문대학 컴퓨터시스템과 교수

<관심분야> : 임베디드시스템, 유비쿼터스 컴퓨팅, 상황인식서비스, 분산웹서비스

임 기 욱(Kee-Wook Rim) 정회원

- 1977년 : 인하대학교 전자공학과(공학사)
- 1987년 : 한양대학교 전자계산학(공학석사)
- 1994년 8월 : 인하대학교 전자계산학(공학박사)
- 1977년 ~ 1983년 : 한국전자기술연구소 선임연구원
- 1983년 ~ 1988년 : 한국전자통신연구소 시스템소프트웨어 연구실장
- 1989년 ~ 1996년 : 한국전자통신연구원 시스템연구부장, 주전산기(타이컴)Ⅲ,Ⅳ 개발사업 책임자
- 1997년 ~ 1999년 : 정보통신연구진흥원 정보기술전문위원
- 2000년 ~ 현재 : 선문대학교 컴퓨터정보학부 교수

<관심분야> : 실시간데이터베이스시스템, 운영체제, 시스템구조

이 정 현(Jung-Hyun Lee) 정회원

- 1977년 : 인하대학교 전자공학과(공학사)
- 1980년 : 인하대학교 전자공학과(공학석사)
- 1988년 : 인하대학교 전자공학과(공학박사)
- 1979년 ~ 1981년 : 한국전자기술 연구소 시스템 연구원
- 1984년 ~ 1989년 : 경기대학교 전자계산학과 교수
- 1989년 1월 ~ 현재 : 인하대학교 컴퓨터공학부 교수

<관심분야> : 자연어처리, HCI, 음성인식, 정보검색, 고성능 컴퓨터구조