韓國 컴퓨터情報學會 2008 제38차 하계학술발표논문집
第16卷 第1號, 2008. 06.

동시사용자 접근이 용이한 데이터베이스 커넥션 풀 아키텍처

김 영 찬*, 이 세 훈**

Database connection pool architecture for interconnections simplicity access

Young-Chan Kim*, Se-Hoon Lee**

요 약

어플리케이션의 데이터베이스 시스템 사용이 증가하면서, 사용자 급증에 따른 데이터베이스 시스템에 동시접속하는 커넥션 처리 방법에 있어 중요성이 부각되고 있다. 이러한 데이터베이스의 커넥션을 효과적으로 처리하기 위해 데이터베이스 커넥션 풀이 도입되었으나, 동시접속이 증가하는 형태에 따른 커넥션 처리는 미비한 실정이다. 이 논문에서는 기존의 커넥션 아키텍처를 개선하여 주 커넥션 풀과 보조 커넥션 풀, 두 커넥션 풀의 연결을 할당하기 위한 커넥션 브로커 관계에 있어 새로운 아키텍처를 정의하였고, 실험을 통하여 기존 데이터베이스 커넥션 풀 구조와 비교해서 100% 가량 성능이 향상된 결과를 제시하였다.

▸ Keyword : 데이터베이스 커넥션 풀(database onnection pool : DBCP), 커넥션 브로커(connection broker), 커넥션 아키텍처(connection architecture), 자바 데이터베이스 연결도(java database connectivity : JDBC)

• 제1저자 : 김영찬
* 케이티하이텔솔루션(주) ** 인하공업전문대학

Ⅰ. 서 론

IT 산업이 발전하고 인터넷이 보급화 됨에 따라 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 어플리케이션들은 데이터베이스를 참조하기 위해 JDBC(Java Database Connectivity)를 사용하게 된다[1]. JDBC드라이버는 데이터베이스에 커넥션을 맺고 어플리케이션의 질의 요청에 따른 결과값을 제공하여 준다.

어플리케이션은 다양한 형태로의 업무 환경을 구성 하게 되는데 데이터베이스 자원을 사용하는 것은 가장 많은 시간을 필요로 하는 부분이다. 즉, 데이터베이스에 연결하여 세션을 얻는 시간이 가장 오래 걸리고 시스템 리소스를 많이 소모하는 부분이다. 어플리케이션의 특성상 연결을 얻고 해제하는 작업은 클라이언트들 마다 각각 빈번하게 발생하게 되고 서버의 입장에서는 무수히 많은 데이터베이스 연결 및 해제 작업을 하게 되므로 오버헤드가 많이 발생하게 된다[3].

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[4]. 한번 생성된 연결은 작업을 마치고 바로 해제하는 것이 아니라 연결을 pool에 보관해 두고 이후 다시 클라이언트의 연결이 요청되면 이를 재사용 하는 방법이다[5]. 그러나 이와 같은 구조 에서도 커넥션 풀에 대한 문제가 다음과 같이 존재하고 있음을 알 수 있다.

커넥션 풀의 증가율은 웹 어플리케이션의 접속 사용자에 비례하며 ,이에 따라 어플리케이션 서버의 리소스가 동시에 증가하게 되는데 이는 EJB와 JSP, SERVLET 등과 같은 어플리케이션들에게도 한정된 자원에서의 리소스 사용으로, 결국 어플리케이션 서버 전체는 리소스 고갈로 인하여 클라이언트들의 요청을 처리하지 못하는 상태가 발생 하게 된다.

실제로 인터넷원서접수등과 같은 동시접속이 일시에 증가하는 형태의 인터넷 서비스는 동시접속자의 수가 폭주함에 따라 서버가 다운이 되는 현상이 빈번히 발생하였다[2]. 이처럼 데이터베이스를 연동하는 어플리케이션에서 데이터베이스 커넥션 관리방법은 전체 시스템의 안전성 및 성능에 밀접한 관련이 있기 때문에 이를 개선하는 일이 중요하기에 이르렀다.

현재의 데이터베이스 커넥션 풀 아키텍처는 어플리케이션 서버 내에서 관리하고 있고 이는 제한된 자원에서 어플리케이션들과 데이터베이스 커넥션 풀이 서로 경쟁관계를 이루고 있기 때문에 이에 대한 개선이 필요하다는 것을 알 수 있다[1].

이 논문에서는 웹 어플리케이션 구축 시 사용하는 커넥션 풀에 대하여 대량의 동시사용자 접근에 대한 성능을 평가하고, 이에 따른 개선된 데이터베이스 커넥션 풀 아키텍처를 제시하고 구현 하는 것을 목표로 한다. 구현된 커넥션 풀 아키텍처는 다양한 부하 조건을 발생하게 하여 그 성능을 측정함으로써, 데이터베이스 커넥션 풀이 안정적으로 요청을 처리할 수 있는 가를 파악한다.

Ⅱ. 관련연구

이 장에서는 JDBC의 전반적인 개요와 아키텍처에 대하여 설명하고, 이를 활용한 커넥션 풀의 연결 아키텍처 대하여 설명한다.

1) JDBC

일반적으로 JDBC API라 함은, 테이블로 이루어진 data를 사용할 수 있도록 하는 Java API를 말한다. JDBC는 실제로 상표 이름이고 흔히 Java Database Connectivity 라는 뜻으로도 사용된다[6]. JDBC는 Java 어플리케이션의 데이터베이스 시스템에 대한 이식성을 높이기 위하여 설계된 것이다. 다시 말해서, JDBC API를 사용한 어플리케이션은 특정 데이터베이스 관리 시스템에 종속되지 않으며, 수정없이 Sybase, Oracle, SQL Server등 다양한 데이터베이스 관리 시스템을 이용할 수 있도록 한다[6]. Java기반의 어플리케이션들은 데이터베이스에 접근하기 위하여 JDBC API를 사용하며, 이는 각각의 벤더업체들에 의해서 구현하고 제공하고 있다.

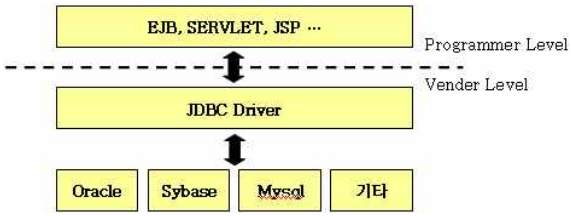
JDBC API는 JDBC 핵심 API와 JDBC 확장 API로 분류된다. JDBC 핵심 API는 어플리케이션이 데이터베이스에 성공적으로 접근하기 위해 반드시 필요한 기능을 지원하는 데 초점을 맞춘 반면, JDBC확장 API는 데이터베이스 접근을 위한 필수 기능은 아니나, 성능과 프로그램 작성의 편리를 위해 유용한 다음과 같은 기능을 지원한다[4].

- JNDI API를 이용한 네이밍 서비스
- JDBC드라이버-기반 연결 풀
- 로우셋(rowsets)
- 분산 트랜젝션

데이터베이스관리시스템 벤더들은 JDBC API에 명세된 대로 자신들의 데이터베이스와 통신할 수 있는 소프트웨어인

JDBC드라이버를 만들어서 배포한다. 이 드라이버는 자바로 만들어진 클래스 파일의 묶음으로 jar나 zip형태로 패키징되어 있다. 이러한 JDBC 드라이버는 JDBC에 정의된 Java명령어를 그들만의 데이터베이스에 맞는 명령어로 바꾸어 전달하고 그 결과로 넘겨 받은 값을 다시 JDBC 드라이버를 통하여 자바 코드에 전달하는 어댑터(adapter)이다[9].

그림 <2-1>은 JDBC 아키텍처를 설명한 그림이다[9]. Java 어플리케이션의 개발은 JDBC 드라이버를 제공하는 모든 데이터베이스를 단일한 인터페이스(JDBC API)로써 접근이 가능하기 때문에 CRUD의 작업이 데이터베이스시스템에 종속적이지 않는 장점을 가지고 있다.



<그림 2-1> JDBC 아키텍처

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

- 2-tier 모델

2-tier모델에서는 자바 애플릿과 어플리케이션이 데이터베이스에 직접 연결된다. 여기서는 해당 데이터베이스 관리 시스템과 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.

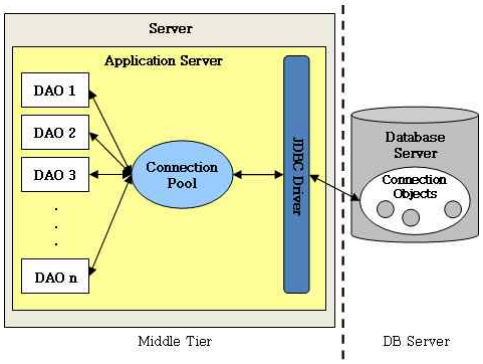
- 3-tier 모델

3-tier 모델의 경우, 2-tier 모델에 미들 tier를 추가한 것으로 SQL문은 미들tier로 보내고, 이는 다시 SQL 문장을 데이터베이스 관리 시스템에 보낸다. 데이터베이스 관리 시스템은 SQL문장을 처리하여 결과를 미들tier에 보내고, 이는 다시 클라이언트에 전달된다. 미들 tier는 엔터프라이즈 환경을 위해 sun에서 제안한 J2EE 스펙을 구현한 어플리케이션 서버를 이용한다[1]. 이러한 어플리케이션 서버에는 'bea사의 Weblogic', 'Tmax사의 JEUS', 'IBM 사의 Websphere' 등이 있다.

2) 커넥션 풀

웹 응용과 같이, 다수의 클라이언트가 빈번히 데이터베이스 접근을 요구하는 환경에서, 데이터베이스 접근을 필요로 하는 모든 클라이언트는 각각 데이터베이스 연결을 생성하고, 이 연결을 이용하여 데이터베이스 작업을 수행한 후 연결을 해제하는 것은 데이터베이스 관리 시스템 입장에서 매우 큰 오버헤드가 되어 성능을 저하시키는 원인이 된다[6].

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 커넥션 풀에 의한 연결 아키텍처는 다음과 같다.



<그림 2-2> 커넥션 풀 연결 아키텍처

이러한 구조는 동시에 여러 클라이언트의 요청을 커넥션 풀에 대기 중인 커넥션으로부터 할당 받고 대기 중인 커넥션이 풀에 없을 경우 추가로 커넥션을 풀에 할당하여 이를 사용하게 된다. 그리고 사용이 끝난 커넥션은 풀에 회수 하여 재사용을 하게 된다. 일반적인 커넥션 풀의 관리 과정은 다음과 같다[8].

가. 커넥션 풀에서 커넥션 얻기
  1) 최초 요청시 일정수의 커넥션을 생성해서 풀에 보관
  2) 클라이언트(어플리케이션)가 커넥션을 요구
  3) 커넥션 제공
  4) 커넥션 사용

나. 커넥션 풀에서 커넥션 반환
  1) 커넥션 반환
  2) 반환된 커넥션을 풀에 저장

다. 커넥션 풀에 의한 이점은 다음과 같다.
  1) 커넥션을 미리 생성해 놓고 요청시 바로 풀로부터 사용함으로써 커넥션 생성시간을 절약할 수 있다.
  2) 최소한의 커넥션을 생성하고 사용함으로써 자원을 효율적으로 사용할 수 있다.

3) 생성된 커넥션은 사용 후에 풀에 보관되어 재사용 된다.

이러한 이점에도 불구하고 근본적으로 구조적인 문제점이 있음을 알 수 있다. 커넥션 풀은 어플리케이션 서버에 종속적이다 라는 것이다. 어플리케이션이 할당하고 있는 자원에 한하여 커넥션을 생성할 수 있다는 것을 알 수 있다. ASP(Application Service Provider) 서비스와 같이 커넥션이 빈번히 발생하는 서비스는 한정된 자원을 효율적으로 활용하여 최대한 많은 커넥션을 풀에 확보하여야 한다.

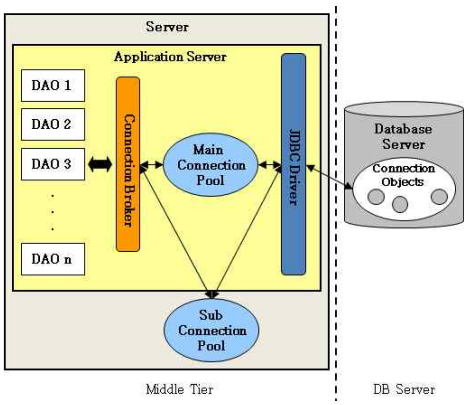
III. 커넥션 풀 아키텍처 설계 및 구현

이 장에서는 기존의 커넥션 풀의 제약된 문제점을 개선하여 한정된 자원을 효율적으로 활용하여 동적으로 커넥션 풀을 연결하기 위한 커넥션 풀 아키텍처를 설계하고, 이를 구현한다.

1) 커넥션 풀 아키텍처 설계

그림 <3-1>은 이 논문에서 설계한 개선된 커넥션 풀 연결 아키텍처이다. EJB나 SERVLET등과 같은 어플리케이션들의 Data Access Object는 커넥션 브로커에게 커넥션할당을 요청하고 커넥션 브로커는 어플리케이션 서버 내의 커넥션 풀로부터 커넥션을 얻어오거나, 서버의 보조 커넥션 풀로부터 커넥션을 얻어오는 구조로 되어 있다. 그리고 이 커넥션 풀들은 JDBC 드라이버를 통하여 해당 데이터베이스관리 시스템의 커넥션을 요청하여 할당 받는다.

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



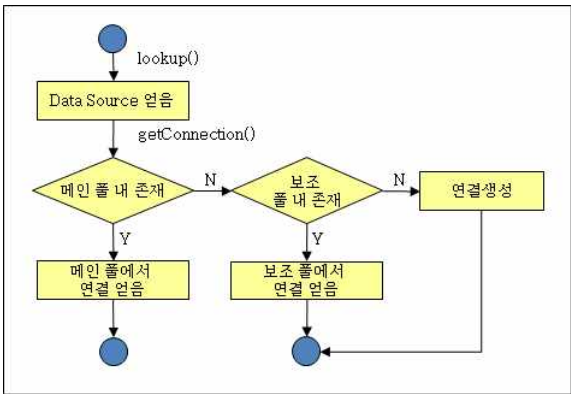
<그림 3-1> 커넥션 풀 연결 아키텍처

2) 시스템 구현

개선된 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 : Xeon 3.2GHz *2, RAM 2GB / HP DL380
- OS : SuSE Linux 9.3
- DBMS : Oracle 10g
- Language : Html, Java Script, Java, Servlet
- JDK : J2sdk1.6
- Web Server : Apache2
- Servlet Engine : Tomcat 6.0

보통 웹 어플리케이션 서버의 구성이 3 Tier인 점을 감안해 3Tier구조로 구현을 하였다. 웹서버는 브라우저로부터 클라이언트의 요청를 받아 서블릿 엔진에 전달하고, 서블릿 엔진은 다시 데이터베이스에게 요청를 전달하고 이를 다시 응답 받는 형태로 구현하였다. 동시사용자 접근이 용이한 데이터베이스 커넥션 풀 구현의 커넥션 브로커의 처리과정은 아래와 같다.



<그림 3-2> 커넥션 브로커 아키텍처

JNDI lookup()에 의해 DataSource를 할당 받고, getConneciton()을 요청 한다. 이때, 커넥션 브로커에서 메인 커넥션 풀에 풀이 존재하면, 이를 할당하고 그렇지 않으면

보조 커넥션 풀로부터 커넥션을 할당한다. 만약 보조 커넥션 풀에서도 미리 생성된 커넥션 풀이 존재 하지 않는다면 직접 커넥션을 생성하여 이를 할당한다. 이에 대한 핵심 알고리즘은 아래와 같다

<코드 3-1> 커넥션 할당 핵심 알고리즘

```
INITIAL DataSource Object FROM Lookup
INITIAL Connection Object FROM getConnction Function
in DataSource
FUNCTION getConnection(Connection)
INITIAL MainPool Object FROM DBCP
INITIAL SubPool Object FROM DBCP

IF (Exist Connection in MainPool)
THEN
GET Connection in MainPool
ELSE
IF (Exist Connction in SubPool)
THEN
GET Connection in SubPool
ELSE
GET Connection in JDBC Driver
END IF
END IF
RETURN Connection
END FUNCTION
```

Ⅳ. 실험 및 평가

이 장에서는 실험방법을 제시하고, 관련연구에서 설명한 데이터베이스 커넥션 풀을 대상으로 그 성능을 평가하고, 3장에서 구현된 동시사용자 접근이 용이한 데이터베이스 커넥션 풀과의 차이점에 대하여 기술한다.

1) 실험 방법

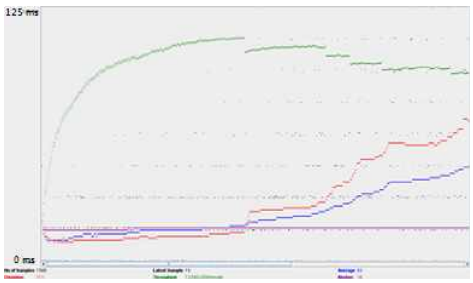
이 논문에서는 데이터베이스 커넥션을 얻어 처리하는 웹 어플리케이션의 처리 시간을 얻기 위하여 동시 접속자를 늘려가면서 그 성능을 측정 하였다. 측정의 대상은 관련연구에서 설명한 데이터베이스 커넥션 풀에 의한 방법과, 이 논문에서 제시한 커넥션 풀의 방법이다.

주 커넥션 풀의 크기는 Default로 설정 되어 있는 20 으로 설정하였고, 보조 커넥션 풀의 크기도 주 커넥션 풀의 크기와 동일하게 설정 하였다.

성능 측정 도구로는 Apache JMeter를 사용 하였으며, 동시접속자의 수는 순간적으로 500명이 접속하는 형태로 3번의 요청을 통하여 총 1500건의 트랜잭션이 이루어지도록 설정 하였다. 이는 사용자의 행위나 데이터의 크기, 유형에 영향을 받지 않는 결과를 도출할 수 있다.

2) 실험 평가

첫 번째로, 데이터베이스 커넥션 풀에 의한 방법에 대한 실험 결과는 다음과 같다.

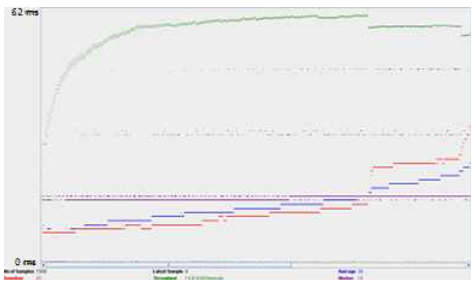


■ Throughput : 12,660.008/min
■ No of Samples : 1500
■ Deviation : 111
■ Average : 53
■ Median : 16
■ Latest Sample : 15

<그림 4-1> 커넥션 풀 연결 방식의 응답시간

그림 <4-1>의 결과 그래프를 보면 처리량이 급속히 증가함에 따라 데이터의 응답시간 분포가 대체적으로 고르게 표시되어 있지만, 처리량 증가에 따라 처리과정의 응답시간은 점점 늦어지고 있다는 것을 알 수 있으며, 최대 125ms까지 응답시간을 기록하였다.

두 번째로, 이 논문에서 제시한 동시사용자 접근이 용이한 데이터베이스 커넥션 풀에 대한 실험 결과는 다음과 같다.



■ Throughput : 14,619.883/min
■ No of Samples : 1500
■ Deviation : 43
■ Average : 30
■ Median : 16
■ Latest Sample : 0

<그림 4-2> 제시한 풀 연결 방식의 응답시간

그림 <4-2>의 결과 그래프를 보게 되면 처리량이 급속히 증가함에 따라 데이터의 응답시간이 62ms 까지 올라가지만 분포의 형태를 살펴보면, 47ms 와 32ms, 16ms의 세 영역에 걸쳐 비교적 많이 표시되어 있는 형태이고, 최대 62ms의 응답시간을 기록하였다.

V. 결론 및 향후 연구

이 논문에서는 동시사용자 접근이 용이한 데이터베이스 커넥션 풀 아키텍처를 제시하고 이를 구현 하였다. 실험 결과를 통하여 기존의 데이터베이스 커넥션 풀 방식은 처리량이 급속히 증가함에 따라 데이터 처리 시간 또한 비례하게 증가하는 반면, 이 논문에서 제시한 풀 연결 방식은 부하가 증가함에 따라 처리 효과가 기존 데이터베이스 커넥션 풀 방식에 비하여, 100% 가량 높아 졌다는 것을 실험을 통하여 확인할 수 있었다.

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, 동시접속이 증가함에 따라 주 커넥션 풀에서 처리용량의 한계점에 올랐을 때, 보조 커넥션 풀이 그 역할을 보조함으로 전체적인 시스템 성능에 있어 처리 용량의 한계를 끌어 올렸다.

이 논문에 이은 향후 연구로는, 다양한 프레임워크에 호수가 가능하도록 데이터베이스 커넥션 연결에 있어, 사용이 동일하도록 프로그래밍 인터페이스를 보완하여야 하며, 주 커넥션 풀과 보조 커넥션 풀 간의 커넥션 할당 알고리즘을 보완하여야 할 것이다.

참고문헌

[1] Java.sun.com, "JDBC Architecture", http://java.sun.com/docs/books/tutorial/jdbc/overview/architecture.html, February 2008.

[2] Joins 뉴스, "CJ 채용 19일 마감 접속자 폭주로 서버 다운", http://article.joins.com/article/article.asp?Total_ID=2889936, 2007.

[3] JavaExchange, "JAVA SERVELET TECHNOLOGIES", http://javaexchange.com, November 2002

[4] Jon Ellis & Linda Ho, "JDBC 3.0 Specification Final Release", Sun Microsystems, Inc., October 2001

[5] Java.sun.com, "Dive into connection pooling with J2EE",http://java.sun.com/developer/technicalArticles/J2EE/pooling/, October 2000.

[6] George Reese, "Database Programming with JDBC and Java", J2EE, O'Reilly, 2000.

[7] Seth White and Mark Hapner, "JDBC 2.1 API, Sun Microsystems", October 1999

[8] 최영관외 3명, "소설 같은 JSP, Jabook", 2002

[9] 천기숙, "JDBC 연결 풀 관리자 설계 및 구현", 성신여자대학교 석사논문, 2002