工學碩士 學位論文

指導敎授 鄭 仁 煥

데이터베이스 연결 Pool 관리를 위한 성능 진단도구의 설계 및 구현

Design and Implementation of Performanece Diagnostic Tool for Database Connection Pool Management

2003 년 6월

한성대학교 컴퓨터 신기술 대학원

컴퓨터 신기술 학과

컴퓨터 공학 전공

李 載 煥

工學碩士 學位論文

指導敎授 鄭 仁 煥

데이터베이스 연결 Pool 관리를 위한 성능 진단도구의 설계 및 구현

Deesign and Implementation of Performance Diagnostic Tool for Database Connection Pool Management

위 論文을 工學碩士 學位論文으로 提出함

2003 년 6월

漢城大學校 컴퓨터 신기술 大學院

컴퓨터 신기술 學科

컴퓨터 工學 專攻

李 載 煥

이재환의 工學碩士 學位論文을 인정함

2003 年 6 月 日

심사위원장 김 일 민 (인)

심사위원 정 인 환 (인)

심사위원 조 세 홍 (인)

목 차

그림 목차

요 약

웹 어플리케이션 개발 시 데이터베이스 시스템의 사용이 증가함에 따라 데이터베이스 시스템에 접속하는 커넥션 리소스 관리에 대한 중요성이 부각되고 있다. 이러한 데이터베이스의 연결을 효과적으로 관리해주는 것이 DBCP이다. 대부분의 상용 서버에는 이런 DBCP가 내장되어 있고, 또한 DBCP를 관리할 수 있는 기능이 내장되어 있다. 하지만, 상용서버를 사용하지 않는 웹 어플리케이션 구축시 DBCP을 자체 제작하여 사용하는 경우가 많다.

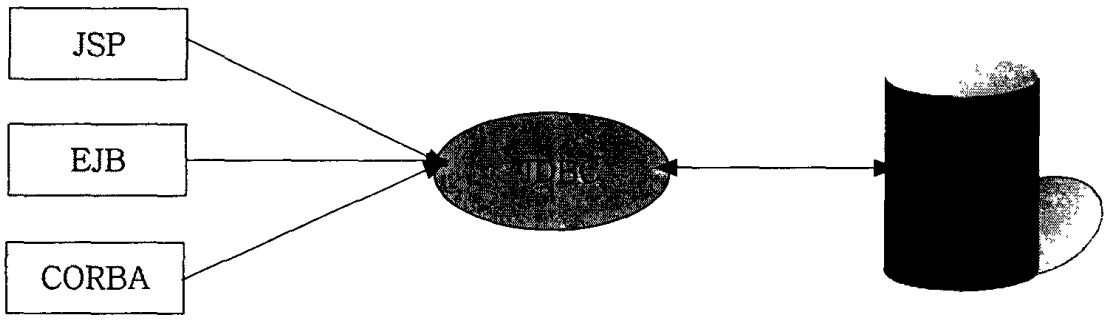
본 논문은 웹 어플리케이션 구축 시 사용하는 DBCP(Database connection Pool)의 성능을 평가하고 진단하는 도구를 제안한다. 본 도구는 DBCP의 성능 및 진단을 통하여 실험에 사용된 DBCP들의 성능과 문제점을 도출하여 DBCP(Database Connection Pool)를 효과적으로 보완할 수 있는 방법을 제시한다.

제 1 장 서론

1.1 연구배경

현대 사회가 정보화 되고 IT산업이 발전하면서 사용하는 데이터의 양이 급격히 증대되고 다양화 되면서 기업체들은 정적인 웹 어플리케이션이 아닌 데이터베이스를 연동한 동적인 웹 어플리케이션 구축의 필요성을 느끼게 되었다. 최근 개발되는 대부분의 웹 어플리케이션들은 데이터베이스 자원을 참조하지 않는 것은 거의 없을 것이다.

최근 들어 가장 많이 활용되는 JSP/SERVLET, EJB[1], 또는CORBA[2]와 같은 JAVA기반의 어플리케이션들은 데이터베이스를 참조하기위해 JDBC(Java Database Connectivity)를 사용하게 된다[1]. [그림 1.1]은 이러한 관계를 보여준다.



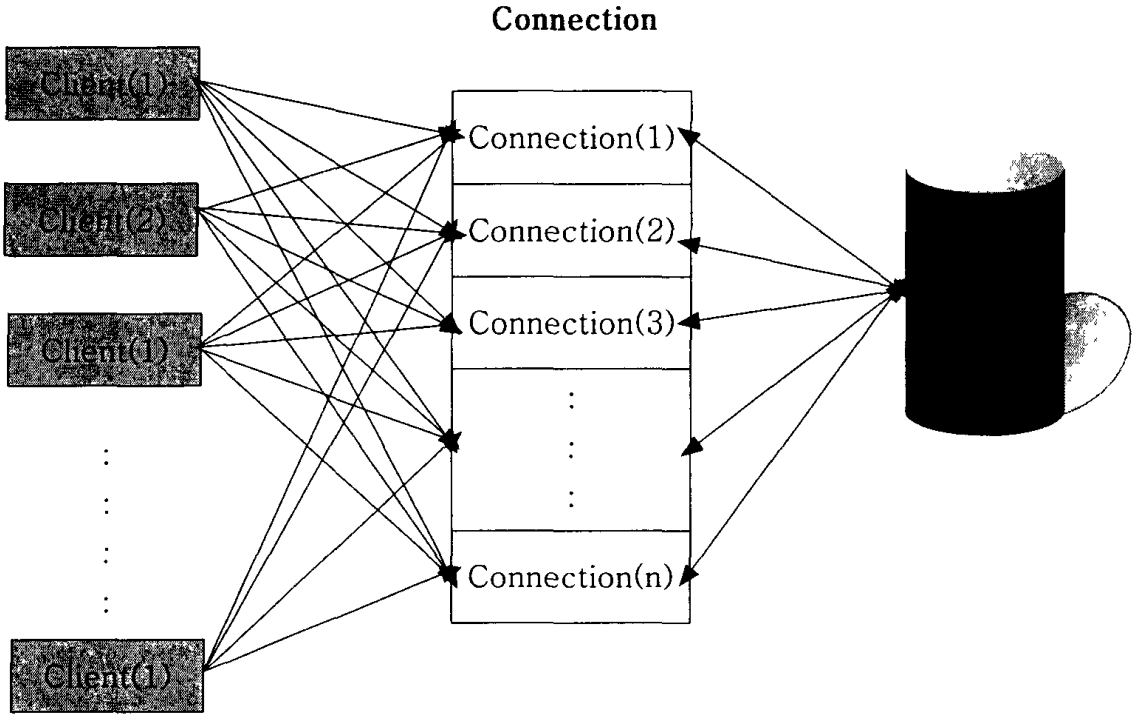
[그림 1.1] 데이터베이스 참조 관계도

JDBC드라이버는 데이타베이스에 커넥션을 맺을 때 사용되는데, 어플리케이션은 이 커넥션 자원을 얻어 데이터베이스에 질의어를 보내고, 질의 결과를 받아 원하는 데이터를 얻는다.

앞에서 설명한 JAVA기반의 어플리케이션 구축시 데이터베이스를 참조하기 위한 커넥션을 많이 맺게 되는데, 웹 어플리케이션의 경우 사용자가 많으면 많을수록 이러한 커넥션을 맺는 일이 많이 일어날 것이다. 그리고, 커넥션은 데이터베이스에서 그수가 제한을 받는데 많은 커넥션이 발생함에도 불구하고 커넥션 관리가 제대로 되지 않으면 전체 시스템이 마비되는 장애를 초래 할 수 있다.

한번 맺은 커넥션은 사용후 반드시 닫아줘야 한다. 왜냐하면 동시에 맺을 수 있는 데이타베이스 커넥션의 수는 항상 데이타베이스에서 제한을 받는데, 제한된 커넥션의 수를 초과하게 되면 그 뒤에 들어온 클라이언트는 더이상 커넥션을 사용할 수 없게 된다.

[그림 1.2]는 DBCP(Database Connection Pool)가 나오기전에 사용하던 미리 커넥션을 맺어놓는 구조다. 이 구조는 메모리상에 데이터베이스 커넥션을 미리 생성해 놓고 요청에 대한 처리를 하므로 속도면에선 향상 될 수 있으나, 가용한 커넥션을 모두 미리 생성해 놓으면 자원의 낭비를 가져올 뿐만 아니라, 동시에 여러 클라이언트들이 커넥션을 요청했을 때 동일한 커넥션을 사용함으로써 발생되는 데이터베이스 잠금(DB Locking)과 거래(Transaction)의 중첩현상이 일어날 수 있다.



[그림 1.2] 커넥션을 미리 맺어놓는 구조

이러한 문제점들을 보완하기 위해 나온 커넥션 관리 기법이 DBCP(Database Connection Pool) 기법이다.

DBCP란 한번 맺은 데이타베이스 커넥션을 바로 끊지 않고 데이타베이스 커넥션 Pool에 저장해 놓고 다음에 동일한 요청을 해오면 바로 Pool에서 저장해둔 커넥션을 꺼내서 줌으로써 빠른 데이타베이스 커넥션 시간을 보장해 주며, 커넥션을 미리 맺어 놓는 것이 아니라 최소한의 커넥션만으로 요청을 처리하므로 자원의 효율을 가져올 수 있다[2]. 이러한 이유로 DB 커넥션 풀의 사용이 증가하고 있으며 최근 대부분의 어플리케이션에서 이 DBCP를 사용하고 있다.

이처럼 데이터베이스를 연동하는 웹 어플리케이션에서 데이터베이스 커넥션 관리는 전체 시스템의 안정성 및 성능에 밀접한 관계가 있다. 웹 기반 어플리케이션의 역사가 짧다 보니, 웹 기반 어플리케이션의 데이터베이스 커

넥션 관리 능력을 어떻게 측정되어지는가에 대한 이론적 지식이 많이 부족하다. 따라서, 이러한 DBCP의 성능 진단 및 분석을 위한 도구를 설계하고 구현하는 일이 중요하다.

1.2 동기

일반적으로 상용 응용서버(Application Server) 제품들은 대부분 DBCP를 제공하고 있다. BEA사의 WebLogic 서버는 Kona JDBC Driver를 이용하여 커넥션 Pool 기능을 제공하며, IBM사의 WebSphere 서버는 자체 제작한 커넥션 Pool 기능을 제공한다. 그러나, 만약 우리가 만들 시스템이 응용 서버(Application Server)를 사용하지 않아 DBCP 기능을 제공 받지 못한다면 어떻게 할것인가?

중소업체나 소규모업체에서는 비싼 응용서버(Application Server)를 구입하지 못하므로 DBCP를 자체 제작하는 경우나 Poolman, JDF DB Connectionpool과 같이 개발 프레임워크에서 라이브러리 형식으로 제공하는 DBCP를 사용하는 사례가 많다. 이러한 경우 다음과 같은 관리 기능이 제대로 수행되는지 점검이 필요하다.

- 사용하지 않거나 죽은 커넥션 자동 커넥션 Pool에 반환하는 기능
- 최대 동시접속수 , 최대 Statement 수 초과시 예외처리하는 기능
- 커넥션수 일정시간 주기로 체크하여 최소 커넥션수로 유지 기능
- 반환되지 않은 커넥션 추적기능

위와 같은 관리 기능이 없거나 미비하다면 효과적인 DBCP 기능을 기대하기 어려울뿐아니라, 데이터베이스 접속이 되지 않아 서버를 재시동해야하는 상황이 발생할 수 있다. 그러므로, 자체 제작한 DBCP의 최적화된 성능 및 안정성을 위해 문제점들을 발견하고 효과적으로 기능을 개선하기 위한 성능 평가와 검증을 위한 별도의 진단 도구를 설계하고 구현하는 일이 중요하다.

1.3 목표

본 논문에서는 웹 어플리케이션 구축 시 사용하는 DBCP(Database Connection Pool)의 성능을 평가하고 진단하는 도구를 구현하여 웹 어플리케이션에 가장 적합한 DBCP를 최적화 하는 방법을 제시하는 것을 목표로 한다.

성능 진단의 과정은 다양한 부하 조건을 발생하게 하여 DBCP(DataBase Connection Pool)를 여러 가지 부하를 발생시켜 성능을 진단함으로써 DBCP가 안정적으로 커넥션 요청을 처리 낼 수 있도록 설계되고 구현되어야 한다.

성능 진단도구는 대부분의 상용 DBMS를 전부 지원하지는 못한다. 따라서 이러한 여러 상용 DBMS를 지원하도록 범용적으로 설계되고 구현되어야 하며, 편리한 사용자 인터페이스(User Interface)를 제공해야 한다, 따라서, 이러한 점을 고려하여 성능 진단도구에 새로운 기능을 추가하는 것이 용이하도록 일반적이고 유연한 구조로 설계되어야 한다.

본 논문의 구성은 다음과 같다. 2장 관련연구에서는 일반적으로 사용하는 데이터베이스 커넥션 모델의 일반적인 구조와 사용 방법에 대하여 설명한다. 3장에서는 DBCP 진단도구의 설계 및 구현에 대하여 설명한다. 4장에서는 실험 및 평가에 대하여 설명한다. 마지막으로5장에서는 결론 및 향후 연구에 대하여 기술한다.

제2장 관련연구

DBCP에 대한 최적화 기법은 개발자들에 의해 지속적으로 연구 및 개발되어 오고 있다. 자체 개발한 커넥션 Pool이나 무료로 제공되는 커넥션 Pool이든 사용하기 전에 해당 프로그램에 사용이 적합한지 먼저 성능 테스트와 기능 분석을 해야한다. 이러한 점을 보완하고 최적화하기 위해 본 논문에서는 실제 웹사이트 개발시에 사용한 DBCP와 무료로 제공되는 DBCP를 대상으로 다양한 형태의 커넥션조건으로 데이타베이스 커넥션을 발생시켜 DBCP의 성능과 문제점을 파악하여 응용 프로그램(Application Program)이 최적화된 성능을 보일 수 있게 할 수 있는 도구를 구현하고자 한다.

2.1 데이터베이스 커넥션의 구조

2.1.1 요청시 마다 커넥션을 맺는 구조

요청시 마다 DB 커넥션을 맺는 구조는 클라이언트의 커넥션 요청시에만 커넥션을 맺는 구조다. 이 구조는 클래스마다 커넥션 객체를 생성하는 코드를 집어넣어야 하는 불필요한 점이 있다. 즉, 클라이언트이 요청시마다 다음과 같은 문장을 실행시키는데 이것이 상당한 부하가 걸린다는 것이다.

```
String driver = "oracle.jdbc.driver.OracleDriver";
String url = "jdbc:oracle:thin:@127.0.0.1:1521:ORCL";
String user = "scott";
String password = "tiger";
Connection Conn = DriverManager.getConnection( dbUrl, user, password );
```

오라클의 경우 이 문장을 실행시에 300 - 500 ms 정도가 걸린다. 이 수치는 작을 듯 하지만 소프트웨어 구조에 따라서 때론 메소드(Method) 단위로 데이터베이스 커넥션을 맺고 메소드가 끝나는 시점에서 커넥션을 닫는데, 한 어플리케이션에서 대략 수백에서 수천개의 메소드에서 커넥션을 맺는다면 이것은 엄청난 부하를 초래한다.

이 구조의 경우 미리 커넥션을 맺어놓지 않기 때문에 자원의 활용면에서는 효과적일 수 있으나 요청시마다 커넥션 객체를 생성해서 커넥션을 맺기 때문에 속도나 성능이 떨어진다.

2.1.2 미리 커넥션을 맺어놓는 구조

이 구조는 앞에서 본 [그림1.2]와 같이 가용한 DB 커넥션을 미리 맺어 놓고 클라이언트의 커넥션 요청에 대해 처리하는 구조로, 예를 들면 서블릿(Servlet)의 경우 커넥션을 init()에서 미리 맺어 두고 doGet()이나 doPost()에서 그 커넥션을 가지고 별도의 Ststement와 ResultSet을 사용하는 것이다. 이러한 경우 세가지의 문제를 야기시킨다.

첫째, 자원의 낭비다. 이 구조는 서블릿당 하나씩 커넥션을 점유하고 있는 셈이 되는데, 보통 실제 웹 프로젝트에서 보통 적게는 100이상은 되며 수백개가 되는 것도 있다. 이럴 경우에 커넥션에 할당 되어야 할 "데이터베이스 연결갯수"도 서블릿 갯수 만큼 필요하게 된다. 즉, 아무런 요청이 없을 때도 수백개의 데이터베이스 커넥션이 연결되어 있어야 한다는 것이다. 이것은 자원의 낭비를 가져온다.

둘째, 대량의 동시 사용자 처리가 어렵다. 만약, 한 서블릿에 대해 동시에 50개 이상의 요청이 들어왔을 경우 같은 커넥션에 대해 50개 이상의 요청이 conn.createStatement()를 호출하게 된다. 이때 문제가 발생하게 되는데 하나의 커넥션에 대해 동시에 열 수 있는 Statement의 개수는 제한을 받는다. Oracle DB의 경우 설정파일에서 커넥션수과 Statement의 최대값을 설정해 놓는다. Ststement의 경우 보통 50인데 , 이 최대치를 넘어서 동시에 Statement를 열려고 하면 " maximum open cursor exceed!" 또는 "결과집합을 모두 소모했습니다."라는 SQLException을 발생하게 된다.

셋째, 거래(Transaction)의 중복현상이 발생한다. 동시에 요청이 들어오게 되면 같은 커넥션을 갖고 작업을 하고 있으므로 Transaction이 중첩되게 된다. 왜냐하면, commit(), rollback()은 같은 커넥션에 대해서 Transaction이 관리되기 때문이다. 즉, 먼저 요청한 A가 작업을 수행하는 중에 B가 들어와 SQL 문장을 수행중에 SQLException을 발생하여 rollback()이 호출된다면 A가 작업했던 것도 같이 rollback()이 되어 버린다.

이렇게 미리 커넥션을 맺는 구조는 커넥션 요청에 대해 속도면에서는 효과적이나, 위와같은 여러 가지 문제점을 안고 있다. 그래서, 이런 문제점을 보완하기 위해서 DBCP라는 기법이 나오게 되었다.

2.1.3 DBCP(Database Connection Pool)를 이용하는 구조

DBCP란 최소한의 커넥션만 미리 맺어놓고 클라이언트의 요청을 처리하고 사용이 끝난 커넥션을 다른 클라이언트에게 재사용하게 하는 구조이다. 이 구조는 앞에서 설명한 두 구조와는 달리 데이터베이스 커넥션을 제공해 주는 별도의 클래스를 통해 커넥션을 유연하게 제공한다.

이 구조는 동시에 여러 클라이언트의 요청이 들어왔을 때 현재 커넥션 Pool에 대기중인 커넥션 수보다 요청수가 많을시에만 커넥션을 추가로 더 생성을 해서 요청에 대해 처리를 한다. 그리고, 사용이 끝난 커넥션은 DBCP에서 회수하여 재사용을 하게 된다. DBCP에서 커넥션을 관리하는 과정은 아래와 같다[3].

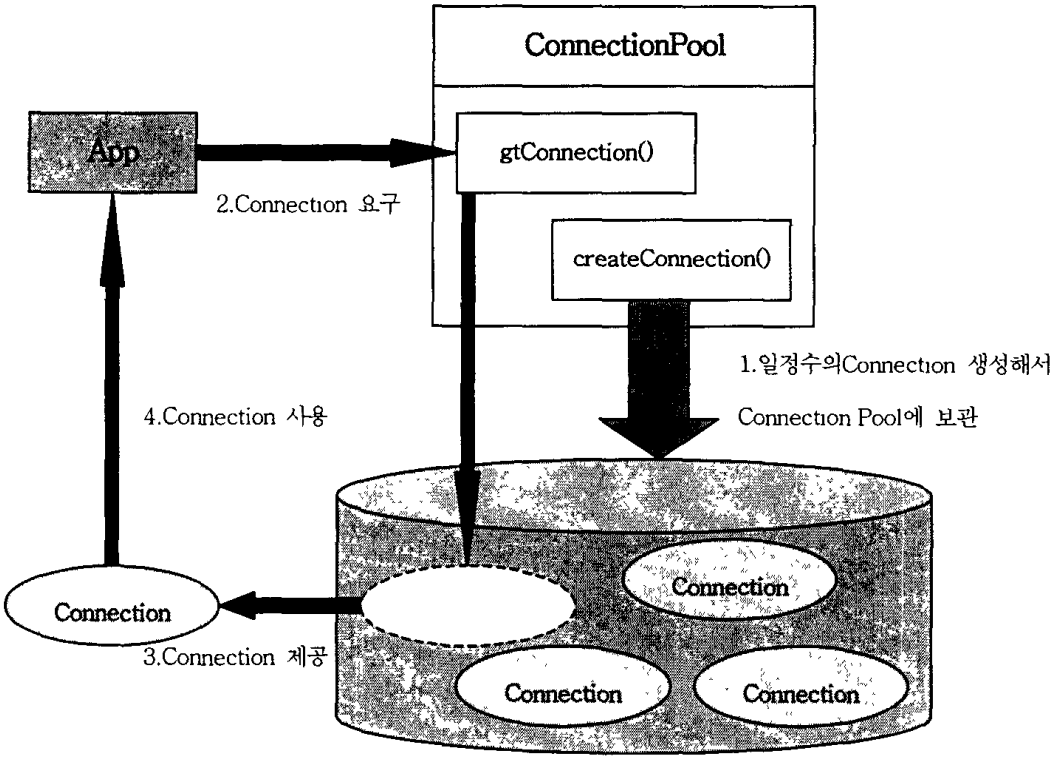
■ 커넥션 Pool에서 커넥션 얻기 [그림2.1]

1) 최초 요청시 일정수의 커넥션을 생성해서 커넥션 Pool에 보관

2) 클라이언트(어플리케이션)가 커넥션을 요구

3) 커넥션 제공

4) 커넥션 사용

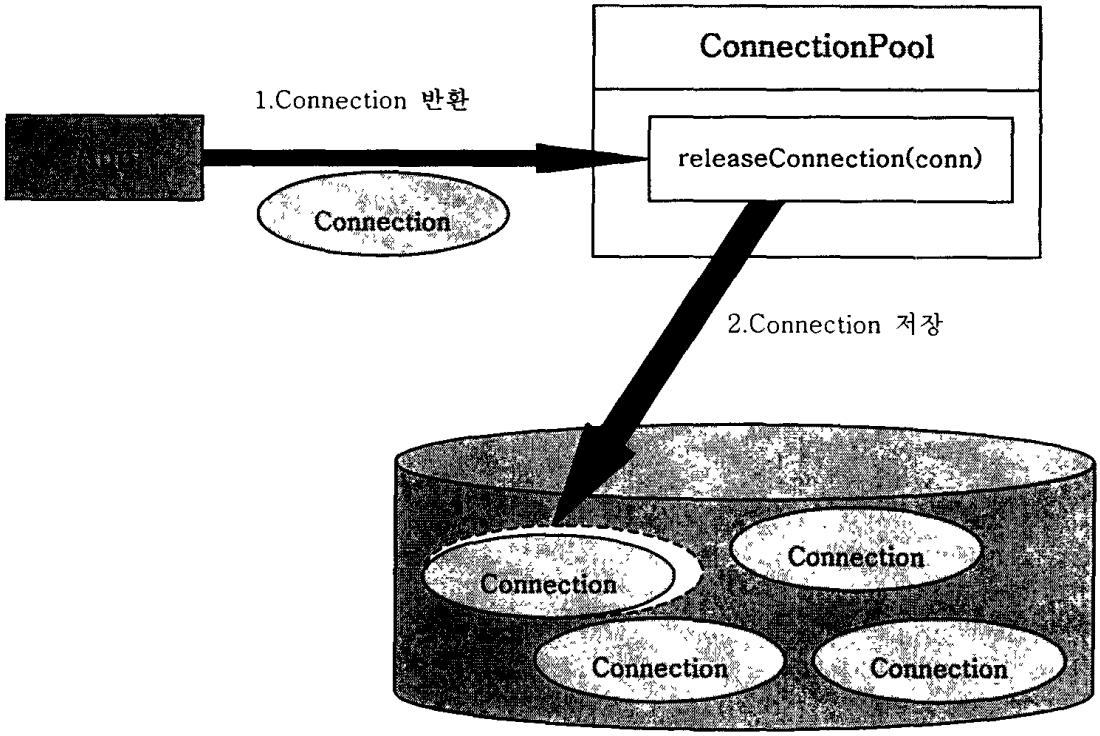


[그림 2.1] 커넥션풀에서 커넥션 얻기

■커넥션 사용후 커넥션 Pool에 반환하기 [그림2.2]

1) 커넥션 반환

2) 반환된 커넥션 Pool에 저장



[그림 2.2] 커넥션풀로 커넥션 반환

위와같이 DB 커넥션을 관리하는 DBCP은 다음과 같은 장점을 가진다.

1) 커넥션을 미리 생성해 놓고 요청시 바로 Pool에서 꺼내줌으로 속도를 향상시킬 수 있다.

2) 최소한의 커넥션만 생성해 놓음으로 자원을 효율적으로 사용할 수 있다.

3) 생성된 커넥션은 사용 후에도 Pool에 보관되어 재사용된다.

DBCP에도 몇가지 문제점은 있다. DBCP가 커넥션 연결 시간을 단축해 주고, 메모리 자원 활용면에서 자원 낭비를 줄여 주지만, 프로그램상에서 개발자가 실수로 커넥션을 사용후 커넥션 Pool에 반환하지 않았을 경우,

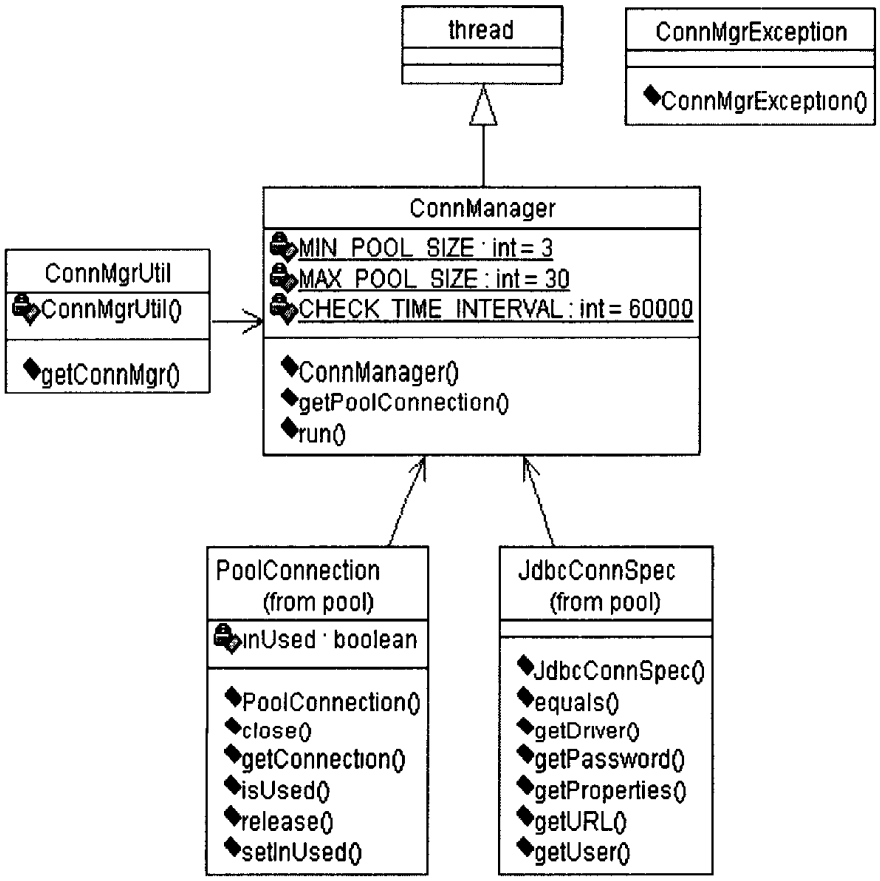
커넥션 최대 연결 개수는 데이터베이스에서 제한을 받게 되는데 이럴 경우 시간이 갈수록 사용중인 커넥션은 증가하고, 가용한 커넥션은 점점 줄어들어 결국에는 어느 시점에서 더 이상 커넥션을 맺을 수 없다는 SQLException이 발생한다.

원칙적으로 프로그램상에서 커넥션을 반드시 닫아줘야 하나, DBCP에서 주기적으로 사용하지 않는 커넥션들을 찾아내 강제로 Pool로 반환하거나 닫아주는 기능이 있다면 동시 접속자가 폭주가 아닌 이상 어느정도 커넥션 관리가 될 것이다.

또 다른문제는 커넥션을 닫아줬다고 해도 Statement를 닫아주지 않았다면 이 Statement는 커넥션의 자원이기 때문에 GC(Garbage Collection)이 되지 않는다. 이럴 경우에도 Statement가 계속 쌓여 결국 SQLException이 발생한다. 그래서 Statement에 대한 관리 기능도 있어야 한다.

2.2 일반적인 DBCP(Database Connection Pool) 모델

다음에 제시한 DBCP는 일반적으로 중소규모의 자바 응용 프로그램(Java Application Program)에서 많이 사용하는 JDF(Java Development Framework) DB Connectio Pool의 모델이다[2].



[그림 2.3] DB Connection Pool Model

이 모델에서는 ConnManager클래스가 스레드(Thread)로 돌면서 DBCP를 관리하고 있다. ConnManager클래스는 Configuration클래스를 이용해서 Pool의 각종 설정을 properties 파일에서 읽어와 커넥션 Pool을 설정한다. 설정파일의 내용은 다음과 같다.

```
# JDF DB Connection Pool
# Connection Pool의 각종 정보를 설정한다.
db.pool.checkinterval = 10000
db.pool.size.min = 3
db.pool.size.max = 30
db.pool.trace = false
```

각각의 설정이 의미하는 것은 다음과 같다.

db.pool.size.min = 3

이 min값은 Pool에 항상 커넥션이 남아 있는 수를 의미한다. 즉, 동시에 여러 Client가 Pool 커넥션을 요청하면 물리적인 데이타베이스 커넥션의 수치가 올라간다.

그러나 모든 클라이언트가 Pool 커넥션을 Pool에 반환하면, Pool에서 대기상태로 남게 되고 이는 다시 다음 요청 시에 재사용될 수 있다. 이 때 이와 같이 재사용 가능한 Pool 커넥션의 갯수를 이 파라메터로 조정할 수 있다. Default는 3이다. 그리고, 처음부터 이 수치만큼 Pool이 생성되어 있지는 않는다. 점차적으로 증가하여 이 수치로 커넥션을 유지 한다.

db.pool.size.max = 30

max값은 Pool에서 동시에 가져갈 수 있는 최대의 Pool 커넥션의 수이다. 즉, 많은 클라이언드들이 Pool에서 커넥션을 가져갔으나 아직 반환이 이루어지지 않았고, 이미 Pool의
최대치에 도달했다면 더 이상 데이터베이스 커넥션을 새로 맺지 않고 ConnMgrException을 발생한다. 이 수치는 물리적으로 데이터베이스 커넥션의 최대치와 밀접한 관련이 있으므로 잘 조절해야 한다.

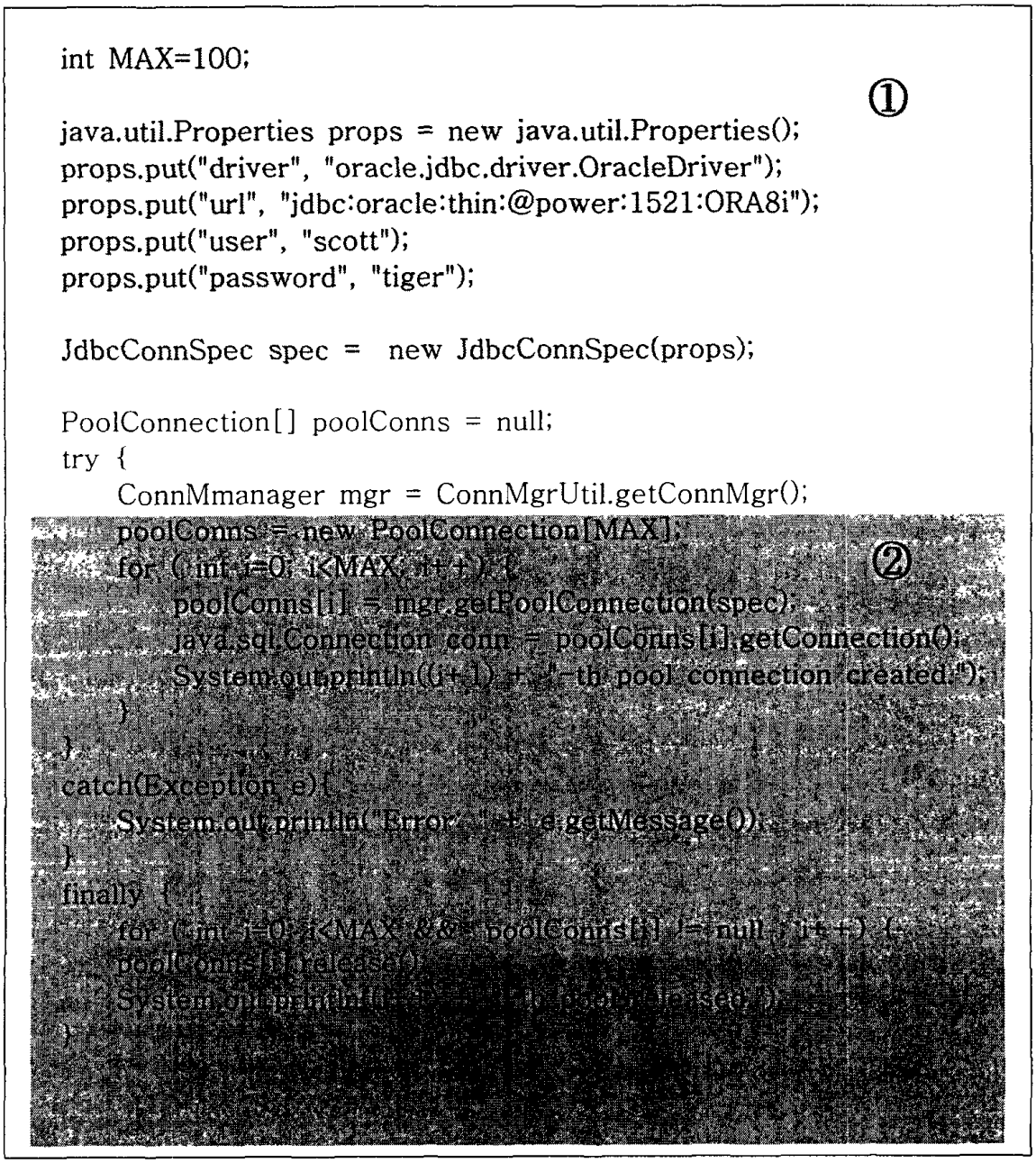
db.pool.checkinterval = 10000

checkinterval값은 ConnMgr가 Pool의 상태를 체크하는 주기다. 즉, ConnMgr가 일정한 주기로 Pool의 상태를 별도의 스레드(Thread)로 항상 체크를 한다. 즉, 현재 가용한 Pool 의 커넥션 수를 db.pool.size.min 갯수로 맞추어 주고, 불필요한 데이타베이스 커넥션은 close() 시킨다. 이와 같이 ConnMgr가 체크하는 시간간격을 나타내는 것으로 1/1000 초
단위의 수치로 표현한다. Default는 10*1000 (10초)이다.

db.pool.trace = false

이 값이 true로 되어 있으면, 일정한 주기마다 현재 Pool의 상태를 로그파일로 남기는
기능을 한다. 또한 이 값이 false일 지라도, Pool에 있는 커넥션의 갯수가 max값보다
더 클 때는 경고성 의미로 이 역시 로그를 남긴다.

다음은 이것을 프로그램에서 사용하는 예이다.



```
int MAX=100;

                                                          ①
java.util.Properties props = new java.util.Properties();
props.put("driver", "oracle.jdbc.driver.OracleDriver");
props.put("url", "jdbc:oracle:thin:@power:1521:ORA8i");
props.put("user", "scott");
props.put("password", "tiger");

JdbcConnSpec spec =  new JdbcConnSpec(props);

PoolConnection[] poolConns = null;
try {
    ConnMmanager mgr = ConnMgrUtil.getConnMgr();
    poolConns = new PoolConnection[MAX];
    for (int i=0; i<MAX; i++) {                           ②
        poolConns[i] = mgr.getPoolConnection(spec);
        java.sql.Connection conn = poolConns[i].getConnection();
        System.out.println((i+1) + "-th pool connection created.");
    }
}
catch(Exception e){
    System.out.println("Error : " + e.getMessage());
}
finally {
    for (int i=0; i<MAX && poolConns[i] != null; i++) {
        poolConns[i].release();
        System.out.println((i+1) + "-th pool released.");
    }
}
```

[그림 2.4] DB Connection Pool 사용예제

이 소스는 PoolConnection을 어떻게 사용하는지 예를 보여주고 있다.

① 부분은 DB 커넥션을 얻기위한 데이터베이스 연결 정보를 설정하는 부분이고,

② 부분은 위에서 연결한 정보를 가지고 실제로 DBCP에서 커넥션을 가져오는 부분이다.

ConnManager 클래스는 스레드(Thread)를 상속받아 별도의 스레드로 동작하는 만큼, System.exit(0)과 같은 명시적인 종료를 하지 않는이상 종료되지 않고 임무를 수행한다. 여기서는 커넥션 Pool 설정을 읽어와 설정값대로 커넥션 Pool의 커넥션을 관리하는 역할을한다. 즉, 커넥션을 스레드로 돌면서 최소 개수로 유지하고, 최대 개수가 초과시에는 ConnMgrException을 발생하며 일정한 시간 주기로 커넥션을 점검한다.

ConnMgrUtil클래스는 ConnManager클래스의 객체를 생성해서 리턴해주는 역할을 하는데 이때, 객체를 요청시마다 매번 생성하는 것이 아니라 싱글턴 패턴을 적용해서 처리를 한다.

ConnMgrException클래스는 예외처리를 담당한다. 즉, 메시지 프로퍼티 파일을 읽어서 해당하는 에러 메시지를 읽어와 뿌려주는 역할을 한다.

JdbcConnSpec클래스는 커넥션을 얻기 위한 데이터베이스 연결정보를 설정하는 클래스이다. 유연하게 동적으로 연결정보를 사용한다면 데이터연결 정보를 프로퍼티 파일로 따로 저장해서 사용하면 된다.

PoolConnection 클래스는 JdbcConnSpec클래스를 이용하여 데이터베이스 연결정보를 설정하고, ConnMgrUtil클래스를 이용하여 ConnManager클래스 객체를 얻어서 커넥션을 얻는 역할을 한다.

2.3 무료로 제공되는 DBCP 분석

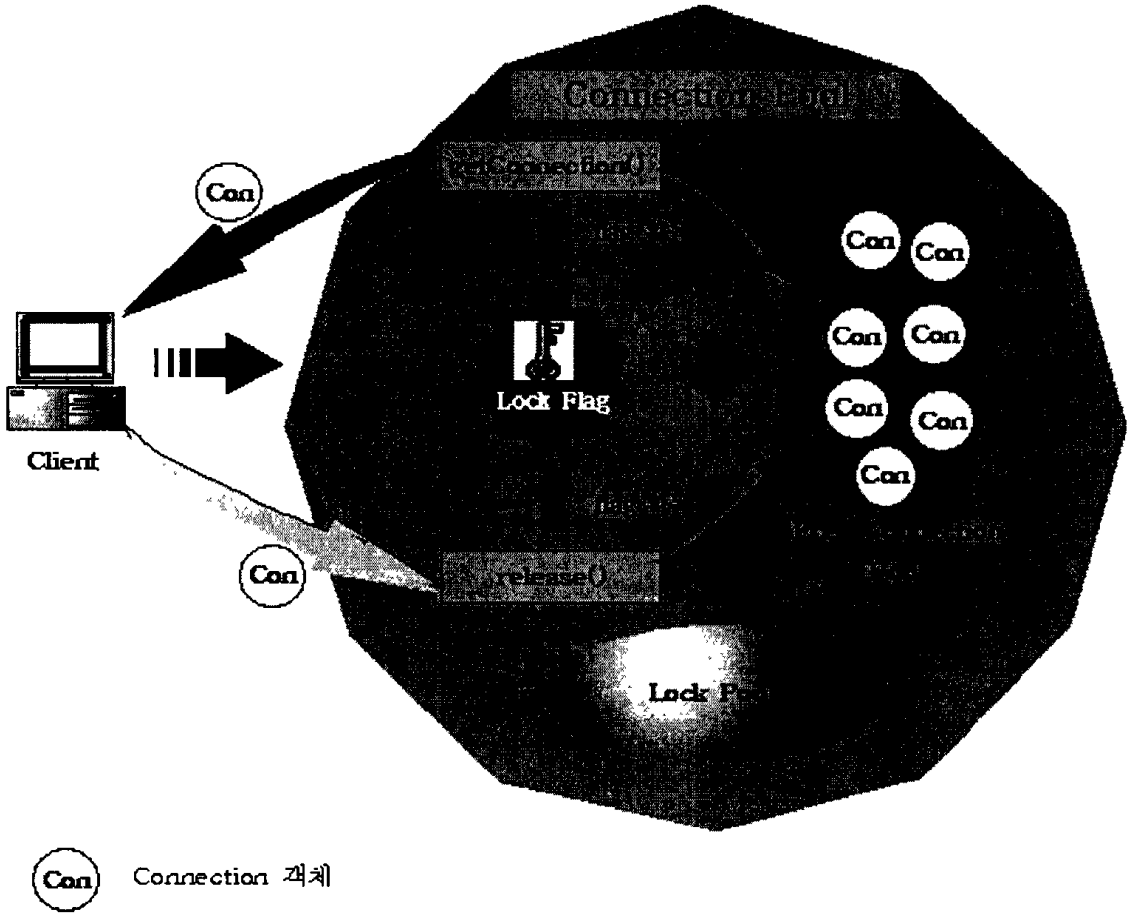
웹상에서 무료로 제공되는 라이브러리 형태의 DBCP들은 많이 있는데, 그중에서 가장 많이 사용하는 DBCP를 대상으로 본 논문에서는 실험을 했다. 다음은 실험에 사용한 DBCP 이다.

- JDF(Java Development Framework)의 DB Connection Pool [2]
- Hans Bergstain의 DBConnectionManager [4]
- Marc A. Mnich의 DBConnectionBroker [5]

이 DBCP들은 전체적인 커넥션 Pool의 구조적 개념은 유사하나, 커넥션 관리 기능면에서 조금씩 상이한 점이 있다.

2.3.1 JDF의 DB Connection Pool

JDF의 DB Connection Pool은 자바 개발 프레임웍의 데이터베이스 연결을 위한 패키지로 4개의 클래스를 제공한다. [그림2.4]는 JDF의 DB Connection Pool의 전체적인 구조를 보여주고 있다.



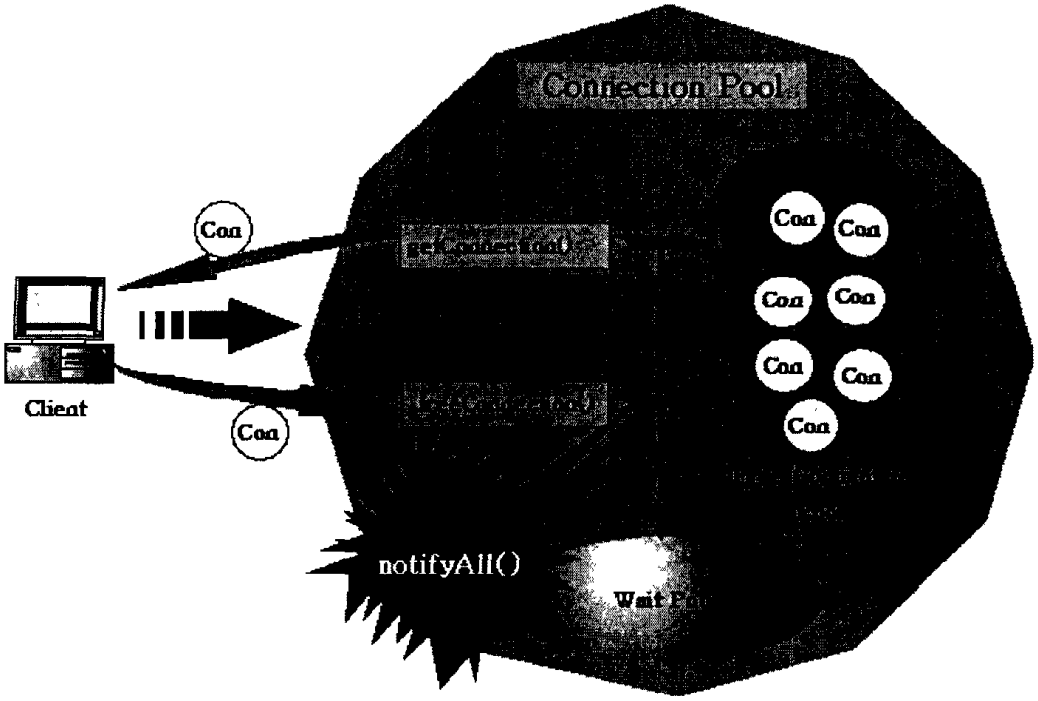
[그림 2.5] JDF Connection Pool 구조

이 구조는 특징은 Connection Pool 객체를 생성시에 싱글턴 패턴을 사용하였으며, 커넥션의 동기화를 위해 getConnection과 release 메소드에 synchronized를 적용했다. 즉, 클라이언트가 데이터베이스를 참조하기 위해 데이터베이스 커넥션을 커넥션 Pool에 요청하였을 때, Locak Flag를 공유해서 순차적으로 커넥션 요청을 처리한다.

Lock Flag는 getConnection과 release 메소드가 공유해서 사용하는데, getConnection이 먼저 Lock Flag를 획득해서 사용하는 중에 다 사용한 커넥션을 누군가 커넥션 Pool에 반환하려고 release를 호출할 때 release는 getConnection이 Lock Flag를 반환할 때까지 Lock Pool에서 대기하게 되는 구조다. 하지만, 커넥션에 대해 동기화하는 것은 좋으나, 이 구조는 Lock Falg 하나를 공유해서 사용하기 때문에 동시에 접속자 수가 폭주했을 때 상황에 따라 release메소드가 사용한 커넥션을 반환하지 못하고 Lock Pool에 마냥 대기하는 현상이 발생해 원활한 커넥션 수급을 할 수 없을 것이다.

2.3.2 Hans Bergstain의 DBConnectionManager

Hans Bergstain의 DBConnectionManager는 비교적 사용방법이 간단하게 구성되어 있으며, JDF DB Connection Pool과는 달리 getConnection과 freeConnection메소드를 동기화하여 커넥션 관리를 하고 있다. [그림2.5]는 전체적인 구조를 보여주고 있다.



[그림 2.6] Hans Bergstain의 DBConnectionManager 구조

Hans Bergstain의 DBConnectionManager의 구조도 JDF와 같이 Connection Pool 객체를 생성시에 싱글턴 패턴을 사용하고 있으며, 커넥션 요청이 적을 때는 Free Connection Pool에 있는 가용한 커넥션으로 처리를 하고, 만약 Free Connection Pool에 가용한 커넥션이 없을 경우에는 wait pool로 들어가 커넥션이 반환될 때까지 기다린다. 그리고, 커넥션이 반환되게 되면 freeConnection메소드에서 반환과 동시에 wait pool에 대기하는 요청들을 notifyAll()을 호출해 커넥션이 반환되었음을 알리는 구조로 되어있다.

이 DBConnectionManager 도 getConnection 메소드를 분석해 보면 심각한 문제점을 안고 있다.

```
public synchronized Connection getConnection() {
        Connection con = null;
        if (freeConnections.size() > 0) { // 현재 가용한 커넥션 수가 0보다크면
            // round_robin 방식으로 Vector를 사용하여
              // 첫 번째 커넥션을 찾아 con에 할당한다.
            con = (Connection) freeConnections.firstElement();
            freeConnections.removeElementAt(0); // Vector 첫 번째 지움
            try {
                if (con.isClosed()) { // 이 con이 사용 불가면 true
                    log("Removed bad connection from " + name);
                    // 다음 커넥션 사용유무 검사위해 다시 호출
                    con = getConnection(); // 및 checkOut 증가
                }
            }
            catch (SQLException e) {
                log("Removed bad connection from " + name);
                // Try again recursively
                con = getConnection(); // 및 checkOut 증가
            }
        }
        else if (maxConn == 0 || checkedOut < maxConn) {
            con = newConnection();
        }
        if (con != null) {
            checkedOut++;
        }
        return con;
    }
```

[그림 2.7] DBConnectionManager의 getConnection 메소드

위에서 ①부분을 보면 현재 가용한 커넥션이 없을 경우 max 값이 0이거나 checkedOut보다 크면 새로 커넥션을 생성한다. 이 경우에는, maxConn 값이 0 일 경우 가용한 Connection 이 freeConnections 에 없다면 무조건 새로운 커넥션을 만들어 주게 된다. 즉, 동시에 과도한 요청을 했을 경우 그만큼의 커넥션이 만들어져서 그 이후에도 계속적으로 연결되어 있는 불합리한 점이 있다.

반면에, 가용한 커넥션이 있을 경우 벡터에서 하나씩 꺼내어 그 커넥션이 현재 사용중인지 아닌지를 판단하게 되는데, 이 경우 위의 소스에는 꺼내온 커넥션이 현재 사용중이면 현재 checkedOut을 증가하지 않고 다시 getConnection 메소드를 호출해야 하는데 이 경우엔 커넥션이 사용중이던 아니던간에 무조건 카운트를 증가시키고 있다. 즉, 이렇게 무한대로 커넥션이 증가하고 maxConn 값이 0 보다 큰 값이 셋팅이 되어 있을 경우에 가용한 커넥션이 없다면 새로 커넥션을 생성해야하는데 이때 이미 카운트가 maxConn값보다 커진다면 무조건 null 값을 반환하게 되어 어플리케이션들이 커넥션을 얻지 못하고 예외상황을 발생하게 된다.

2.3.3 Marc A. Mnich DBConnectionBroker

Marc A. Mnich의 DBConnectionBroker는 Hans Bergstain의 DBConnectionManager의 구조와 매우 유사하다. DBConnectionBroker는 커넥션 카운트 값을 커넥션 배열에서 꺼내온 커넥션이 사용가능 할 때만 증가시키게 했다. 그리고, DBConnectionManager가 가용한 커넥션이 없을때 null값을 리턴하는 것과는 달리 가용한 커넥션이 없을땐 스레드당 2초를 기다리고 10번까지 시도하게 하였고, 20초에 한번씩 커넥션 GC(Garbage Collection)가 운용되도록하여 커넥션이 사용한지 얼마나 지났는지, 커넥션이 사용중인지를 체크하여 지정한 시간이 지났거나 사용중이지 않으면 강제로 닫아주도록 설계하여 안정성을 높였다.

제3장 DBCP 진단도구의 설계 및 구현

3.1 기능 요구 사항 분석 설계

본 논문에서 제시하는 DBCP 성능 진단 도구에서는 다음과 같이 크게 3가지 부분으로 구성되어있다.

■데이터베이스 연결
■부하 발생
■로그 분석

먼저 데이타베이스 연결 구현부에서는 테스트에 사용할 데이터베이스에 대한 연결 정보를 쉽게 설정할 수 있는 인터페이스를 제공한다.
세부적으로는, 데이터에비스 종류, 데이터베이스 이름, 데이터베이스 주소, 사용자 아이디, 사용자 비밀번호, 포트, 테스트에 사용할 DBCP로 구성되어진다.

부하 발생 구현부에서는 데이터베이스 연결 정보로 연결된 데이터베이스에 대해 부하를 발생할 수 있는 항목으로 구성되어 있다. 세부 항목은 다음과 같다.

■ 클라이언트 수(Client Count)
■ 커넥션 발생 간격(Connection Interval)
■ 커넥션유지시간(Connection Time)
■ 동시 커넥션 발생수(Simultaneity Connection)
■ 최소 커넥션수(Minimum Connection)

■ 최대 커넥션수(Maximum Connection)
■ 미반환 커넥션수(Loss Connection)
■ 미반환 Statement수(Loss Statement)

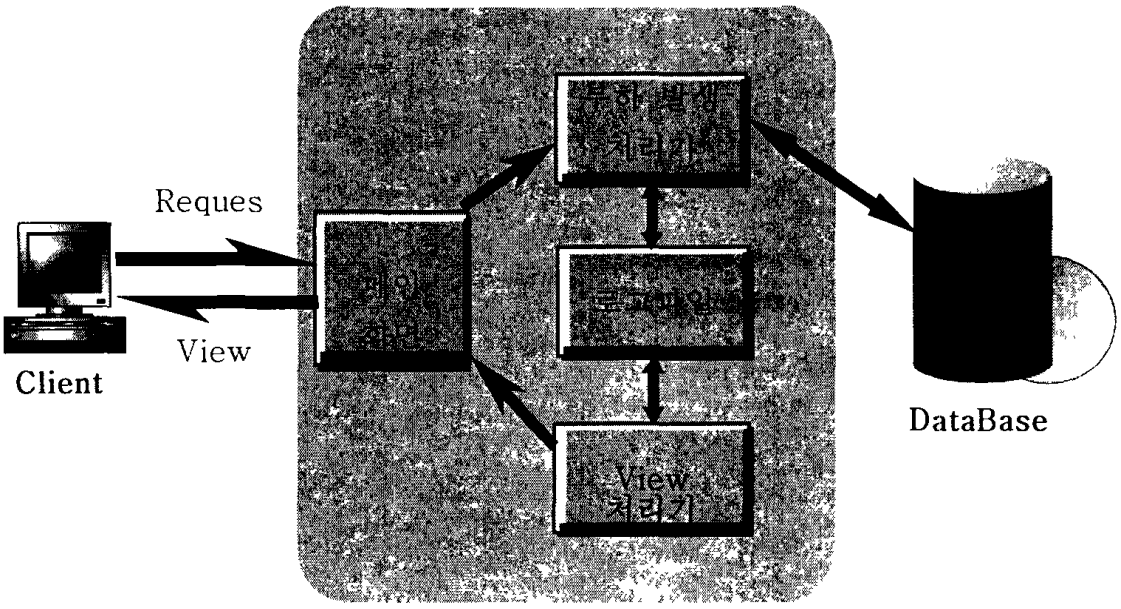
클라이언트 수(Client Count)는 실제로 사이트에 접속하는 사용자수를 말한다. 커넥션 발생간격(Connection Interval)은 실제로 DB 커넥션을 발생시키는 시간 간격이다. 즉, 페이지보기의 클릭하는 간격이라고 할 수 있다.

커넥션 유지시간(Connection Time)은 커넥션을 생성해서 완전히 죽을때까지의 시간이다. 동시 커넥션 발생수(Simultaneity Connection)는 여러 사용자가 동시에 커넥션을 발생시킬 수 있는 최대치이다. 최소커넥션수(Minimum Connection)는 커넥션 풀에서 아무 액션이 없을때 기본적으로 유지하고 있는 커넥션의 개수를 의미하며, 최대 커넥션수(Maximum Connection)는 데이터베이스에서 제한을 받는데 최대로 생성가능한 커넥션 수를 말한다. 미반환 커넥션수(Loss Connection)는 실제로 웹어플리케이션에서 몇 개의 커넥션이 누수하는지 개수를 의미하며, 미반환 Statement수(Loss Statement)는 한 커넥션당 몇 개의 Statement를 누수하는지 개수를 의미한다.

부하 발생 결과에 따른 로그 분석 구현부에서는 3가지 기능을 제공하고 있는데, 로그 View, 명세 출력 , 그래프 View 으로 구성되어 있다. 로그 View는 부하가 발생한 결과가 기록된 로그 파일을 열어 부하가 진행된 히스토리 데이터를 보여주고, 명세 출력은 로그 데이터를 가공하여

Excel 파일로 전환하여 볼 수 있는 기능을 제공하며, 그래프 View는 로그 데이터를 분석 및 가공하여 그래프 형식으로 부하 발생 결과를 보여주는 기능을 제공한다.

3.2 데이터 처리 설계



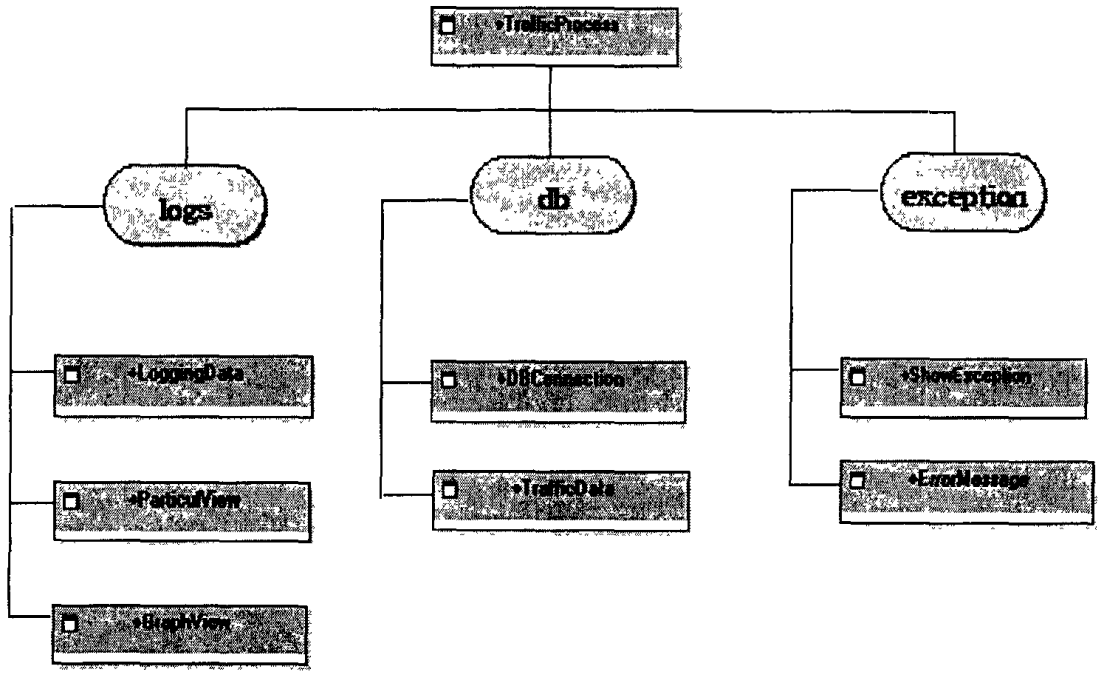
[그림 3.1] 진단 도구의 DPF(Data Process Flow)

그림[2.7]에서와 같이 부하 발생 데이터와 로그 데이터의 흐름은 클라이언트로 입력받은 정보를 받아 부하발생 처리기에서 데이터베이스를 연동하여 부하발생을 처리하고 로그를 로그파일에 기록을한다.

View 처리기는 로그 파일에 기록된 로그 데이터를 분석 및 가공하여 클라이언트에게 다양한 형태로 로그 데이터를 보여준다.

3.3 클래스 설계

본 논문에서 구현한 진단 도구는 총 3개의 패키지와 8개의 클래스 5개의 HTML 파일로 구성되어 있다. logs 패키지는 로그기록과 로그분석을 통한 View 기능을 담당하며, db 패키지는 데이터베이스 연결 정보, 부하발생 데이터를 처리하는 기능을 담당하고, exception 패키지는 부하발생이나 로그분석시 발생하는 각종 예외상황에 대한 메시지 로그 기능을 담당한다. 그 세부적인 클래스 계층 구조는 [그림 2.8]과 같다.



[그림 3.2] 클래스 계층 구조

■ TrafficProcess 클래스 : 실제적인 부하 발생을 담당하는 클래스.
■ LoggingData 클래스 : 부하발생시 발생하는 각종 로그를 기록하는 클래스.
■ ParticulView 클래스 : 로그데이타를 명세표로 출력하게 해주는 클래스.

■ GraphView 클래스 : 로그데이타를 분석하여 그래프 형식으로 보여주는 클래스.

■ DBConnection 클래스 : 설정 데이터를 받아 DBCP의 종류와 각종 데이터베이스 연결정보를 가지고 커넥션을 연결하는 기능을 하는 클래스.

■ TrafficData 클래스 : 메인화면에서 설정한 데이터베이스 연결정보, 부하 발생 조건등의 데이터를 설정하는 클래스.

■ ShowException 클래스 : 사용자 정의 예외발생 클래스.

■ ErrorMessage 클래스 : 각종 상황에 따른 에러 메시지를 출력해주는 클래스.

3.4 구현 환경

3.4.1 구현 시스템 환경

진단 도구 구현에 사용된 웹 서버는 Sun Solaris 8 spark 기종을 사용하였다. 개발툴로는 IMIT사의 Jcradle을 사용하였고, 개발 언어로는 HTML, Java Script, Java, Servlet을 사용하였다. 그리고, HTML과 Java를 분리하기 위해 SourceForge의 FreeMarker를 사용하였다. 본 시스템의 구현 환경을 살펴보면 다음과 같다.

■ System : Pentium III 750 MHZ / Sun Solaris 8 Spark

■ OS : Windows 2000 Server / Unix

■ Database : Oracle

■ Language : HTML, Java Script, JAva, Servlet

■ Library : FreeMarker 1.7

■ JDK : J2sdk1.4

■ Web Server : Apache 1.3.1

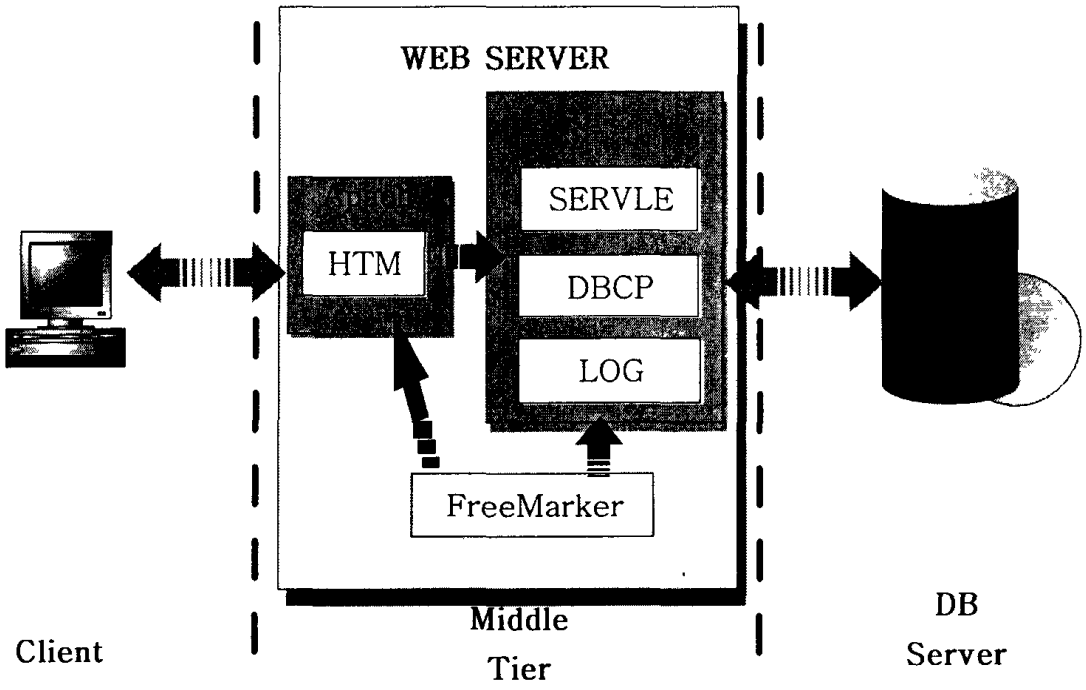
■ Servlet Engine : Resin 2.0

3.4.2 구현 원리

[그림3.3]에서와 같이 본 진단 도구의 구현원리는 보통 웹 어플리케이션의 시스템의 구성이 3 Tier인 점을 감안해 3 Tier 구조로 구현을 하였다.

웹서버는 브라우저로부터 클라이언트의 정보를 받아 서블릿에 받은 데이터를 전달하고, 서블릿은 전달 받은 데이터를 바탕으로 데이터베이스와 연동하여 부하를 발생시키고 로그 파일에 기록을 한다.

로그 분석한 결과를 보여줄때 Html에 Java 코드를 넣지 않고 가독성을 높이고, JSP에서와 같이 Custom Tag 기능을 사용할 수 있게 하기 위하여 FreeMarker 라이브러리를 사용하였으며, 즉 서블릿은 로그 데이터를 가공하여 가공한 데이터를 리스트 형식으로 담아 HTML에 전달해서 보여주는 원리로 구현되었다.



[그림 3.3] 구현 원리

클라이언트는 웹서버에 접속해 진단도구의 메인화면을 요구하게되고 Apache서버는 메인화면을 불러와 클라이언트에게 보여주게 된다. 클라이언트가 데이터베이스 연결정보와 부하발생항목을 입력하고 실행을 하면, Apache는 Resin 서버에게 정보를 넘기고, 해당 Servlet(TrafficProcess class)이 호출되어 넘겨받은 정보(데이타베이스 연결정보, 부하발생정보)를 가지고 DBCP를 연동하여 부하를 발생하고 로깅(LoggingData calss) 클래스를 호출하여 로그파일에 부하발생 데이터를 기록하게된다.

부하발생이 완료한후 View 기능을 호출하면 해당 Servlet(ParticulView class,GraphView class)이 호출되어 로그파일의 데이터를 파싱(parsing)하여 명세표 및 그래프 형태로 보여준다.

Freemarker는 View 기능 Servlet에서 파싱한 데이터를 SimpleHash 형태로 담아 Html 형식으로 변환하여 보여주는 역할을 한다.

3.4.3 구현 알고리즘

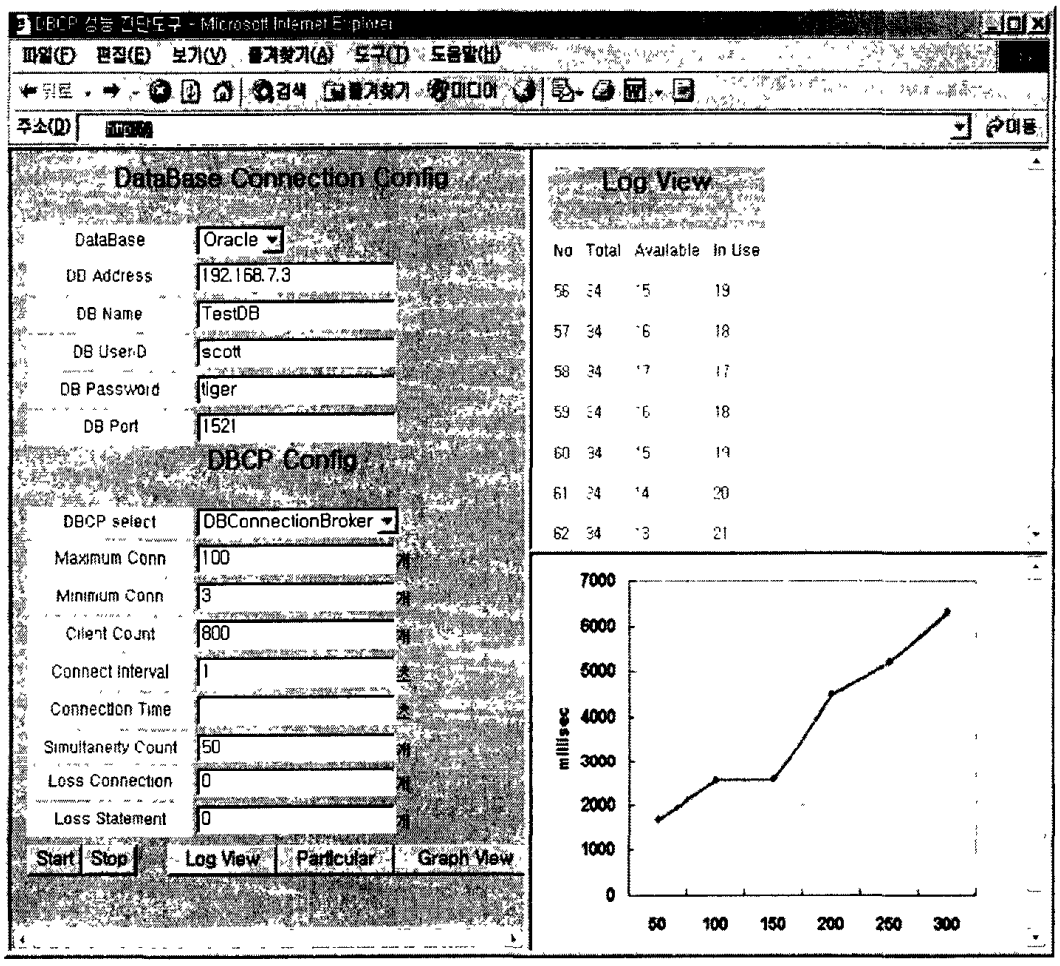
본 연구에서 구현한 진단 도구의 부하 조건에 따른 커넥션을 발생시키는 핵심 알고리즘이다. 부하 발생 데이터는 db 패키지의 TrafficData클래스에 set()메소드로 할당한다.

```
// 커넥션 풀을 초기화 한다
pool = new DBConnection(dbDriver, dbServer, dbLogin, dbPassword, minConns,
            maxConns, logFileString, maxConnTime, DBCP_value);
            :
public void loopTraffic(){
    // 클라이언트수 접속.
    for( count = 0 ; count < TrafficData.getConnCount() ; ){
        // 동시 접속
        for(int i = 0 ; i < TrafficData.getSimConn() ; i++){
            String threadName = count + "번째";
            new Thread(name).start();
            count++;
        }
        try{  // 접속 간격
            Thread.sleep(TrafficData.getConnIntervalTime());
        }catch(InterruptedException e) {
             e.printStackTrace();
        }catch(Exception e){
             e.printStackTrace(),
        }
    }
}
            :
public void run(){
    Date current = new Date();
    long time = 0;
    try{
        time = current.getSeconds(); // 커넥션 만들기전 시간
        log.println(1, getName() , time);
        conn = pool.getConnection();// 커넥션 얻음
        log.println(1, getName() , time);
        if( count%TrafficData.getLossConn() == 0){ // 커넥션 미반환
            ++lossConnCnt;
            log.println(2, lossConnCnt , getName(), time);
        } else { // 정상적인 커넥션 반환
            pool.release(conn);
            log.println(3, getName(), time);
        }
     } catch(InterruptedException e) {
            e.printStackTrace();
     }catch(Exception e){
            e.printStackTrace();
     }
}
```

[그림 3.4] 핵심 알고리즘 코드

3.4.4 진단 도구의 실행 화면

[그림3.5]에서 보듯이 이 진단 도구는 크게 3개의 프레임으로 구분되어있다. 상단 좌측화면이 메인화면이며 오른쪽 상단이 로그 View 화면, 하단 화면이 그래프 View 화면이다.



[그림 3.5] 진단 도구의 실행 화면

[Log View] 버튼 기능은 부하발생이 완료한후 기록된 로그파일을 읽어 StringTokenizer로 데이터를 정재하여 Freemarker의 SimpleHash에 List형태로 담아 오른쪽 상단 프레임에서 보는거와 같이 데이터를 뿌려주는 기능을한다.

[Particuler] 버튼 기능은 [Log View]에서 정재된 데이터를 Excel 파일로 옮겨주는 기능을한다.

[Graph View] 버튼 기능은 [Log View]에서 정재된 데이터를 이용하여 그래프 형태로 보여주는 기능을 한다.

제4장 실험 및 평가

4.1 실험 방법

실험은 2장에서 설명한 3개의 DBCP을 대상으로 그성능을 비교하여 문제점을 도출해 내도록 한다. 구현된 개발 도구를 사용하여 본 논문에서는 3가지 실험을 하였다.

첫 번째, 각 DBCP들의 데이터베이스 커넥션을 얻기 위한 평균 시간을 동시 접속자를 늘려가면서 성능을 테스트하였다.

둘째, DBCP의 GC(Garbage Collection) 성능 평가다. 주기적으로 커넥션을 닫지 않음으로 해서 커넥션이 점점 쌓이는 것을 GC가 잘 처리하는지 테스트하였다.

셋째, Statement의 처리 테스트이다. Statement를 열고 닫지 않은 상태로 커넥션을 닫아 버리면 이 Ststement는 쓰지 않는 고아 상태가 된다. 데이터베이스에서는 이 숫자가 제한을 두고 있는데, 이 Statement 처리 능력을 평가하였다.

4.2 실험 결과 및 분석

4.2.1 실험 1 - DBCP의 데이터베이스 커넥션을 얻기 위한 평균 시간 (속도 평가)

커넥션(Connection) 객체를 경쟁적으로 얻기 위해서 스레드(Thread)의 수

를 여러 개 사용하여 실시간으로 커넥션(Connection) 객체를 얻기 위한 조건 하에서 측정하였다.

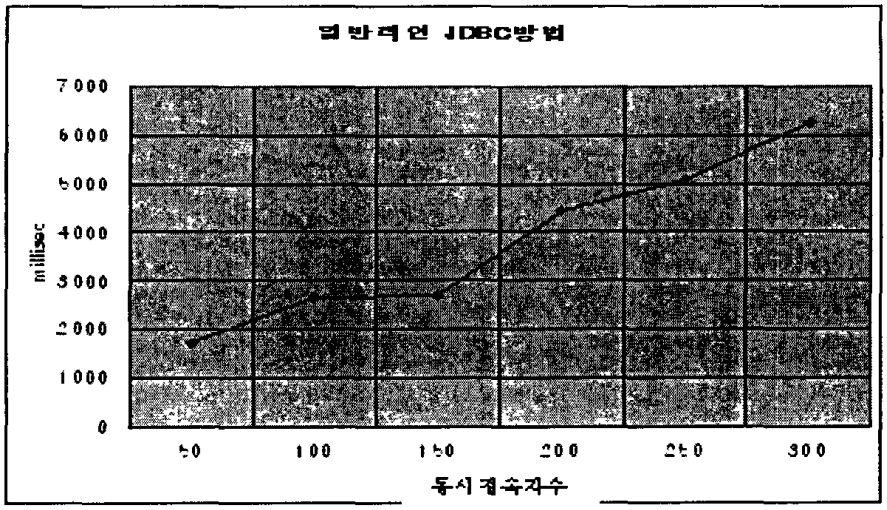
이는 웹 환경에서 클라이언트의 수가 결국 커넥션(Connection)을 얻기 위한 경쟁적인 상태를 유지하는 것이므로 이를 웹 어플리케이션 모델로 구현하여 테스트를 실시하였다.

여기서 제외한 요소는 보통 웹 어플리케이션 서버(Web Aplication Server)에서 servlet/jsp는 다중 쓰레드닝으로 동작하므로 서버의 설정 중 이를 제한하는 옵션을 가지고 있다. 그러므로 실질적인 서버에서의 동작은 쓰레드의 증가 시에 속도 저하 요인이 더 존재 할것이므로 실제는 쓰레드 증가 시(만약 다른 조건들이 똑 같다면) 더 느리게 동작할 것이다.

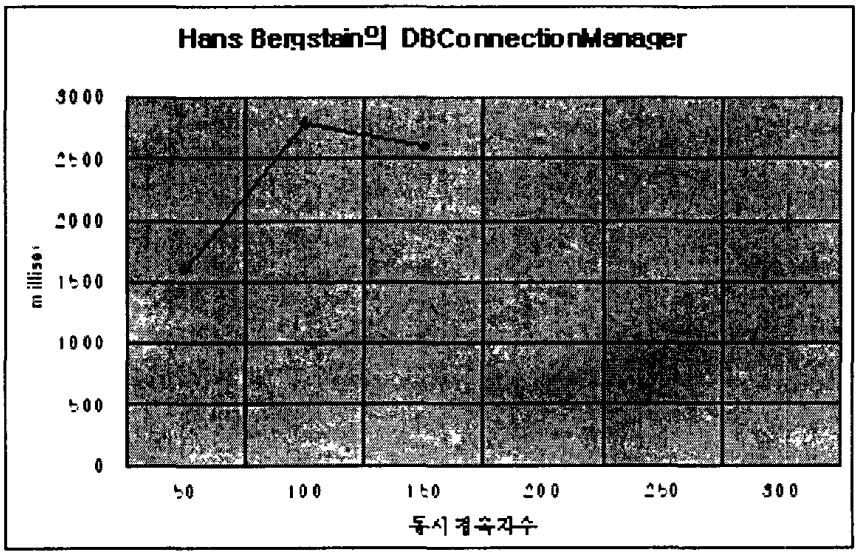
커넥션 객체를 얻어 오는데 걸리는 시간은 보통 적게는 300ms에서 많게는 3초까지 걸린다. 이 실험에서는 커넥션 객체를 얻는 시간의 성능을 측정과 동시 접속자를 처리하는 능력을 측정하였다.

다음은 각 DBCP의 실험 결과화면이다. 참고로, DBCP를 사용하지 않고 직접 객체를 생성해서 얻는 방법을 사용하였을 때의 경우를 추가 실험하였다. 부하조건은 다음과 같다.

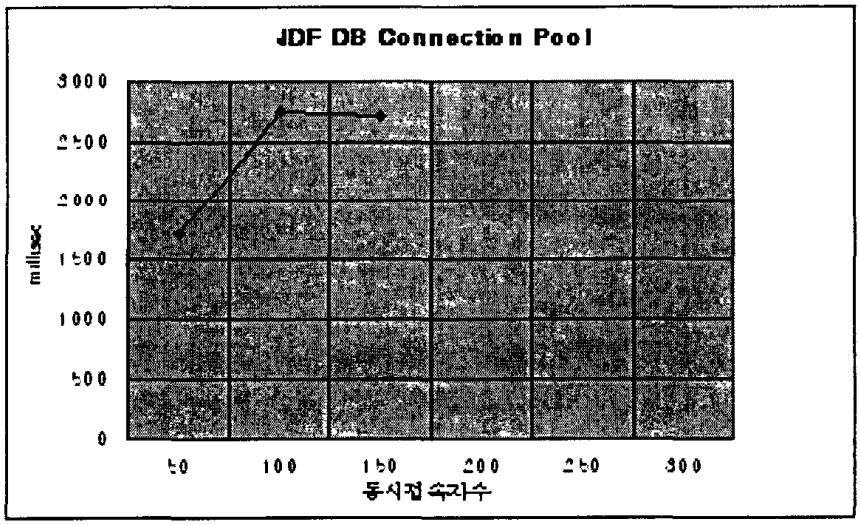
■ 최소 커넥션 / 최대 커넥션 : 10개 / 100개
■ 동시 접속수 : 50개씩 증가



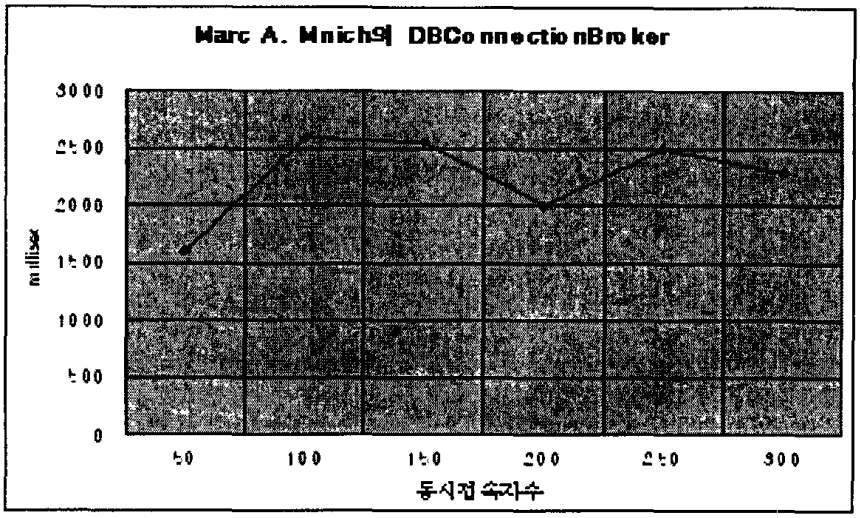
[그림 4.1] 일반적인 JDBC의 커넥션을 얻기 위한 평균 시간



[그림 4.2] JDF의 커넥션을 얻기 위한 평균 시간



[그림 4.3] Hans Bergstain의 커넥션을 얻기 위한 평균 시간



[그림 4.4] Marc A. Mnich의 커넥션을 얻기 위한 평균 시간

[그림4.1]과 같이 일반적인 JDBC API의 방법에 의한 연결은 그 자체로써는 거의 이론적인 결과 수치 를 나타내어서 실질적인 구현 할 때는 현실과는 거리가 먼 것으로 보인다.
50개의 클라이언트 접속을 보일 때는 1초에서 2초의 사이의 성능을 보여 주지만 스레드(클라이언트) 의 수가 증가 할 수록 실험 오차를 생각 하더라도 거의 일차 방정식의 형태로 진행되고 있는 것을 볼 수 있다. 이는 클라이언트의 수가 많을 수록 커넥션을 얻기 위해서는 더 많은 시간이 걸림을 알 수 있다. 실제 프로젝트에서 이런 성능을 가지고 Data Driven Site가 유지 될 수 없을 것이다.

그러므로, 결국 실험에서 보듯이 Connection Pooling이라는 기술에 관심을 가질 필요가 있다.

[그림4.2]의 JDF 의 DB Connection Pool과 [그림4.3]의 Hans Berstain 경우 비슷한 실험 결과를 보이는데 이 두가지 DBCP는 좋지 못한 Performance를 보여 주고 있다. 테스트 결과 도표에서 보는 것처럼 실시간으로 커넥션(connection)을 얻기 위한 접속 클라이언트의 수가 50일때 부터 100을 지나 150의 동시 접속을 보이기 시작하자 커넥션(Connection) 객체를 얻지 못하는 스레드 (클라이언트)가 발생하기 시작했다. 이는 커넥션(Connection)을 효율

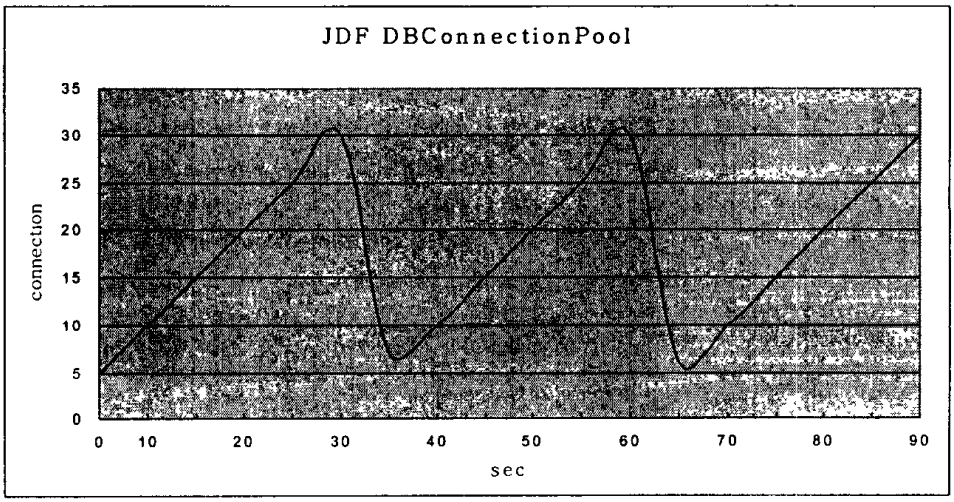
적으로 관리하는 구조를 DBCP가 가지고 있지 않기 때문에 이 DBCP는 동시 접속 클라이언트가 최대 연결 가능 커넥션 수를 초과시에는 초과 요청 클라이언트에 대해 관리를 할 수 있 구조적인 설계가 이루어져야 한다.

[그림4.4]의 DBConnectionBroker는 좋은 Performance를 보여 주고 있다 클라이언트의 수가 50개일때와 100개일때의 수치 증가 이후는 큰 변화 없이 클라이언트의 수가 300일때까지의 성능을 나타내고 있다. 처음 클라이언트가 50에서 부터 100일때의 급격한 수치 증가는 이 DBCP의 커넥션 객체의 최소와 최대 수에 기인한다. 즉 여기서는 최소 10, 최대 100으로 설정했기 때문에 클라이언트들이 커넥션 객체를 요구하는 수가 클라이언트의 수에 따라 증가함에 따라 미리 DBCP에 담고 있던 커넥션 객체의 부족으로 인해 DBCP는 클라이언트의 요구에 따라 커넥션 객체를 새롭게 만들어서 DBCP에 등록시킨다.

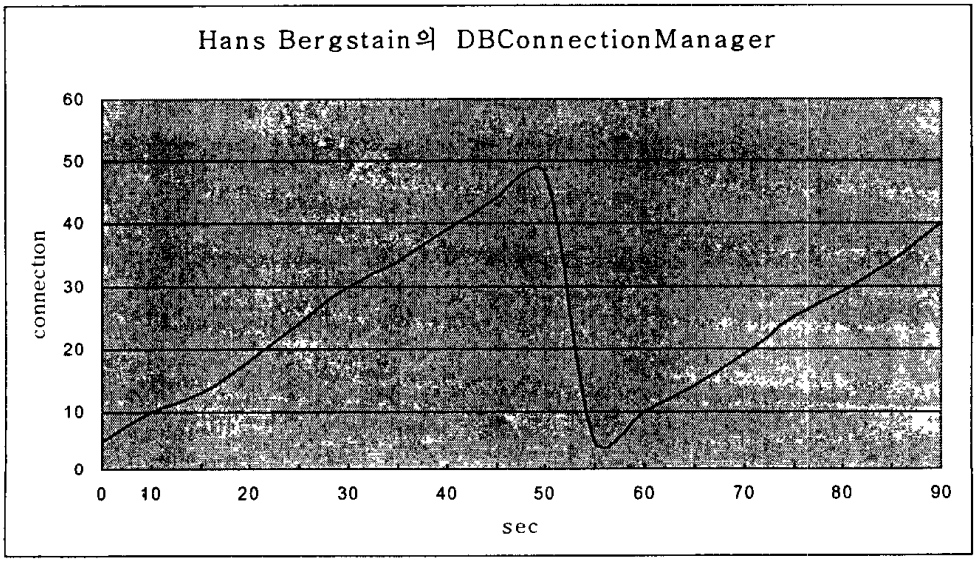
4.2.2 실험 2 - DBCP의 커넥션에 대한 GC(Garbage Collection) 성능 평가

두 번째 실험에서는 DB 커넥션을 얻어서 사용하고 DBCP에 정상적으로 반환된 커넥션을 관리하는 성능을 테스트한다. 부하는 다음과 같다.

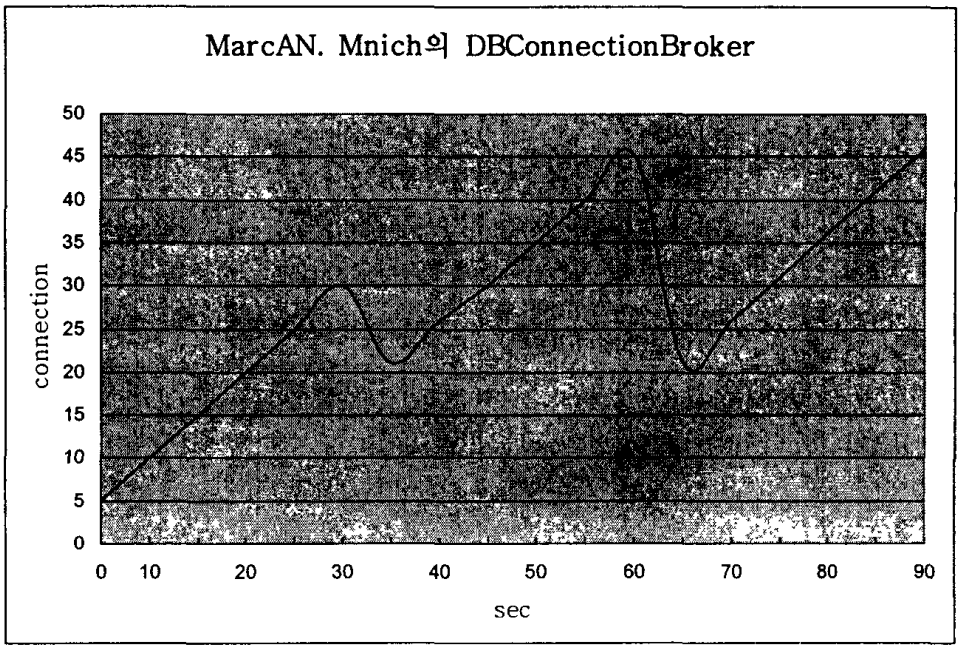
- 발생 커넥션수 : 1개 / 초
- 커넥션 GC Check Interval : 30초
- 최소 커넥션 / 최대 커넥션 : 5개 / 60개



[그림 4.5] JDF의 커넥션에 대한 GC 성능 결과



[그림 4.6] Hans Bergstain의 커넥션에 대한 GC 성능 결과



[그림 4.7] Marc A. Mnich의 커넥션에 대한 GC 성능 결과

[그림4.5]에서와 같이 JDF의 커넥션에 대한 GC 성능 결과로 보아 GC이 30초가 되었을 때 커넥션수가 급격하게 떨어지는 것을 볼 수 있다. 이것은 반환된 커넥션들을 사용중인지 아닌지 검사한 후 사용중이 아닌 것은 모두 강제로 닫고 최소 커넥션수로 낮춰 놓는다. 이것은 자원의 효율성은 있지만, 바로 동시 접속자가 많이 들어왔을때 다시 커넥션을 생성해야하는 부하가 발생한다.

[그림4.6]은 Hans Bergstain의 커넥션에 대한 GC 성능 결과인데, 이 DBCP는 반환된 커넥션이 최대치까지 쌓여도 아무런 기능을 하지 않고 최대치를 초과 했을시 커넥션을 최소치로 낮춘다. 이 DBCP는 항상 커넥션 자원이 Pool에 대기중이므로 커넥션 수급 속도면에서는 효율적일 수 있으나, 커넥션 자원이 항상 연결되어 있으므로 자원낭비를 초래한다.

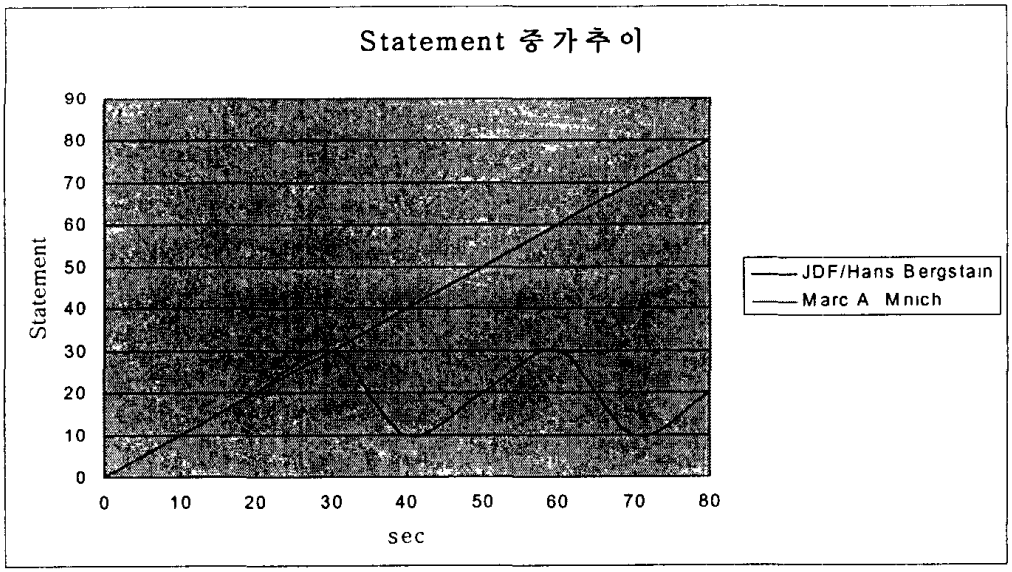
[그림4.7]은 Marc A. Mnich의 커넥션에 대한 GC 성능 결과이다. 여기에서 보듯이 30-35초 사이에 커넥션 수가 급격히 줄지 않고 소량의 수가 줄어들었다. 그리고, 그 수를 계속 유지하는 것을 볼 수 있다. 즉, 이 DBCP는 GC이 커넥션을 최소치로 낮추는게 아니라, 커넥션이 생성된 시간과 현재 시간을

빼서 나온 시간이 20초를 초과하면 커넥션을 닫고 아니면 커넥션 Pool에 남겨두었다. 이 DBCP는 Hans Bergstain의 DBCP와는 달리 커넥션을 다 닫지 않고 커넥션의 시간을 체크하여 다소 여유있는 커넥션으로 다음 커넥션 요청이 많을시 다시 생성하지 않고 빠른 커넥션을 공급할 수 있어 효율적이다.

4.2.3 실험3 - DBCP의 Statement에 대한 GC(Garbage Collection) 성능 평가

보통 커넥션을 닫으면 Statement도 닫힐것이라고 생각되지만 그렇지가 않다. Statement도 한 커넥션에서 열수 있는 수가 제한이 있다. 그 수를 초과하면 시스템이 다운되는 현상을 초래한다. 이 실험에서는 Statement에 대한 GC의 성능을 평가하고자 한다. 부하조건은 다음과 같다.

- 커넥션 발생수 : 1개 / 초
- 미반환 Statement 수 : 커넥션 1개당 1개
- 최대 열수 있는 Statement 수 : 50개/커넥션



[그림 4.8] Statement에 대한 GC 성능 결과

[그림4.8]에서 보듯이 JDF DBCP와 Hans Bergstain의 DBCP는 Statement의 수가 시간이 갈수록 계속 증가하는 것으로 보아 Statement에 대한 GC 기능이 없는 것으로 판단되며, 이 Statement들은 없어지지 않고 메모리 영역을 차지하여 굉장한 자원 낭비가 된다. 또한, 한 커넥션에 대해 열린 Statemtnt가 50개가 넘으면 "maximum open cursor exceed!" 라는 에러를 발생하고 시스템은 다운된다.

반면 Marc A. Mnich의 DBCP는 GC가 30초 주기로 사용하지 않는 Statement를 모두 강제로 닫아주고 있다. 하지만 이 DBCP의 Statement를 관리하기 위해서는 프로그램에서 별도의 코딩을 필요로한다. 즉, 생성시에 2차원 배열에 커넥션과 Statement를 넣고, 닫을시에도 배열에서 그것을 삭제해줘야 하는 번거로움이 있다.

제5장 결론 및 향후 연구

일반 중소규모업체에서 비싼 응용 서버를 구입할 수 없거나, 해당 프로젝트가 응용서버를 필요하지 않을 경우에 부득이하게 개발자가 자체적으로 무료로 제공되는 DBCP를 사용하거나 이것을 참조하여 자체 개발해서 할 경우가 있는데, 실제 프로젝트에 적용해 높은 성능을 기대하기는 쉽지 않으며, 이러한 DBCP를 최적화하기 위해서는 많은 노력과 지식이 필요하다.

본 논문에서는 이러한 DBCP들의 성능을 진단하여 최적화된 성능과 안정성을 검증할 수 있는 성능 분석 도구를 설계하고 구현하였다.

실험에서는 실 프로젝트에서 가장 많이 사용하고 있는 DBCP를 대상으로 성능 진단도구를 사용하여 실험을 하였다. 실험 결과에서 보듯이 DBCP 성능 진단도구로 무료로 제공되는 DBCP들의 성을을 평가하는 기준인 평균 커넥션 얻는 시간, 커넥션 GC의 성능, Statement 관리등의 기능을 테스트한 결과 각 DBCP들의 좋은점과 문제점들을 효과적으로 도출해 내는 것을 보여주었다.

첫 번째 실험에서는 DBCP를 사용하지 않고 일반적인 JDBC방법으로 커넥션을 맺을시에 동시 접속수가 증가할수록 커넥션을 얻는데 걸리는 평균시간이 일차방정식처럼 증가하는 것을 보았다. 하지만 DBCP를 사용할 경우에도 일부 DBCP에서 커넥션을 얻는데 걸리는 시간이 100 개정도의 동시접속자까지는 어느정도 안정된 평균시간을 보이지만 100개 이상의 동시접속에서는 예외사황(Exception)이 발생해 서비스를 하지 못하는 상황이 발견되었다.즉, 일부 DBCP에서 동시 접속자수의 폭주에 대비한 구조적인 개선이 필요한 것을 볼 수 있다.

두 번째 실험에서는 GC의 성능을 평가하였는데 실험결과에서 보듯이 대부분 GC이 잘 작동되고 있는 것을 볼 수 있었으나, 다소 성능상의 차이를 보여주고 있다. 여기서 Hans Bergstain의 DBCP가 커넥션을 바로 모두 닫지 않고 시간을 체크하여 점차 줄이는 방법을 사용하여 동시접속자의 폭주에 대비에 유연한 구조로 되어있는 것을 알 수 있다.

세 번째 실험에서는 Statement에 대한 GC성능을 평가하였는데 대부분의 DBCP들이 기능이 없거나 있어도 개발시에 코드에 직접 코딩해야하는 의존적인 구조로 되어있어 이것에 대한 기능 개선이 필요하다.

실험을 통하여 각 DBCP들의 성능 결과를 토대로 성능상의 장단점을 파악할 수 있었으며, 어떤 부분을 보완하여 실 프로젝트에 사용해야할지 방법을 제시하였다.

본 논문에 이은 향후 연구로는, 다양한 조건의 DB Connection 부하가 생성될 수 있도록 부하 생성 알고리즘을 보완하고, 로그데이타를 실시간으로 볼 수 있고, 다양한 각도로 성능을 분석 할 수 있도록 로그 데이타를 구성하고, 실제 데이터베이스 프로젝트에 적용할 수 있도록 사용자 인터페이스를 보완하는 것이다.

참 고 문 헌

[1] M. Sood, Examming JDBC Drivers, *Dr. Dobbs Journal*, pp.82-87, Jan 1998.

[2] 이원영, "Java Development Framework Document DB Connection Pool", http://www.javaservice.net, 1999.

[3] 최영관외 3명, 소설같은 JSP, Jabook, 2002

[4] Hans Bergstain, DBConnectionManage, htt://www.webdevelopersjournal.com , 1998

[5] Marc A. Mnich, DBConnectionBroker, http://www.javaexchange.com, 2002

[6] ED Roman, "Mastering Enterprise JavaBeans & the Java 2 Platform, Enterprise ED", Wiley & Sons, 1999.

[7] Object Management Group, Realtime CORBA Joint Re-vised Submission, OMG Document orhos/99-02-12. ed. Mar. 1999

[8] D.S. Brahme, et, al, "Transaction Based Verification Methology," Cadence Berkely Labs, Technical Report, Aug. 2000.

[9] Poolman, http://sourceforge.net

[10] Mike phi, Using JDB in a multithreaded qui environment, http://forum.java.sun.com, 2003.

[11] P. Oneil, Database Principles Programming Performance, Morghan Kaufmann Publishers, 1994

[12] 볼랜드 자바팀,JbuilderStudyNet 공저, Jbuilder7, 가남사, 2003.

ABSTRACT

Design and Imoplementation of Performance Diagnostic Tool for Database Connection Pool Management

Lee, Jae-Hwan

Computer New Technology
of the Graduated School
Hansung University

advised by Jung, In-Hwan

As the use of Databse system increases when developing Web Application, the control of DB connection resource which connects to Database system has become more important. And DBCP(DataBase Connection Pool) is what controls this Database connection effectively. Most Application server has this DBCP built-in and the functions which controls this DBCP. But in case of Web Application which does not use Application Server or of a small size company which can not buy the expensive Application Server, mostly they make their own DBCP and use it. This paper proposes a tool which judges and diagnoses the performance of DBCP used when building Web Application. This tool finds the problems and performance of the DBCP used based on the experiment result through GC(Garbage Collection) performance and

diagnosis, Connection Resource control and simultaneously-connected person management of DBCP, and presents the way to supplement effectively the DBCP.