Object Pooling기법을 이용한 웹 검색 시스템의 성능개선에 관한 실험적 연구

Empirical study on Web-based Search System using Object Pooling Method

김한기[0], 여일연, 윤희준, 윤화묵
한국과학기술정보연구원
Kim, Han-Gi[0], Yeo, il-yeon, Yoon, Hee-Jun, Yoon, Hwa-Mook
Korea Institute of Science and Technology Information

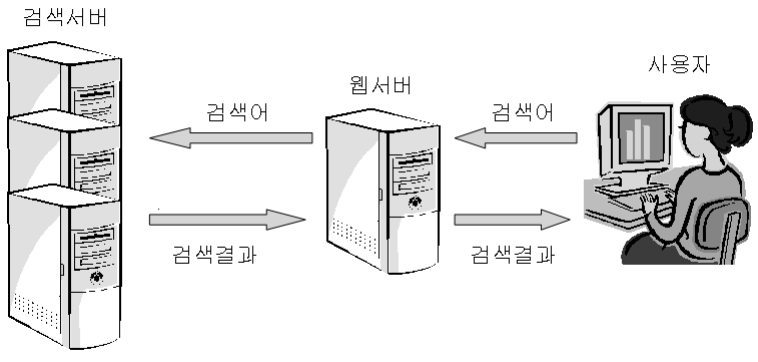
< 요 약 >

일반적으로 웹 검색 시스템은 서비스 서버와 검색 서버가 분리된 구조로 구성되어 있다. 검색 요청이 많아지면 서비스 서버에서 검색 서버로 연결하는 검색 클라이언트 객체를 만드는 작업이 전체 시스템 성능에 부하를 줄 수 있다. 이럴 경우 한번 만들어진 검색 클라이언트 객체를 재사용하면 전반적인 시스템 성능의 향상을 얻을 수 있다. 이는 검색 클라이언트 객체를 재사용하는데 사용되는 비용이 새로운 객체를 생성하는데 필요한 시간과 사용된 객체의 가비지 컬렉션 시간의 합보다 더 작기 때문이다. 본 논문에서는 KRISTAL-2000을 사용한 웹 검색 시스템에 대해서 Object Pooling기법을 적용하여 어느 정도의 성능 향상을 얻을 수 있는지 알아보았다.

키워드 : Object Pool, 웹 검색 시스템, KRISTAL 검색엔진, 검색 클라이언트

1. 서론

일반적으로 웹 검색 시스템은 <그림 1>과 같은 구성을 가진다. 웹 인터페이스를 통해서 사용자가 입력한 질의는 검색 서버로 보내진다. 검색 서버는 해당 질의에 대한 검색을 수행하여 검색결과를 웹 서버로 보내주며, 이 검색결과는 웹 인터페이스를 통해서 사용자에게 제공된다.



<그림 1> 웹 검색 시스템 구성도

이 때 웹 서버와 검색 서버 사이에는 검색 서버와의 통신을 담당하는 검색 클라이언트들이 만들어지게 된다. 검색 클라이언트는 질의어 하나에 대해서 하나씩 만들어지며, 검색 서버로부터 받은 검색 결과를 웹 서버에게

전달해 준 후 사라진다. 요즘 많은 웹 사이트들이 자바(Java)기반의 서블릿(Servlet)이나 JSP로 만들어지기 때문에 검색 클라이언트도 자바 객체로 만들어지는 경우가 일반적이다. 처음 검색 클라이언트 객체가 만들어 질 때에는 검색 서버와 통신 연결을(보안이 필요한 경우에는 로그인 작업이 필요할 수도 있다) 해야 하기 때문에 상대적으로 긴 시간이 필요하다. 또한 생성된 검색 클라이언트 객체는 일반적으로 검색이 끝난 후에 재사용되지 않고 소멸되므로 클라이언트 객체에 대한 가비지 컬렉션(Garbage Collection) 작업에도 일정량의 시간이 소모된다. 검색 시스템과 비슷한 시스템 구성을 가지는 데이터베이스 연동 웹 어플리케이션(Web Application)의 경우에도 데이터베이스 서버와 연동을 담당하는 클라이언트 객체는 이와 비슷한 성격을 가진다. 이 경우 일반적으로 웹 어플리케이션의 성능 향상을 위해서 데이터베이스 커넥션 풀(Database Connection Pool)을 사용하는 것이 일반적인 상황이다.[1]

본 논문에서는 데이터베이스 커넥션 풀과 동일한 역할을 하는 검색 클라이언트 객체 풀을 만들어 웹 검색 시스템에 적용 하였을 때 얼마만큼의 성능향상을 얻을 수 있는지를 실험적으로 알아보았다.

본 논문의 구성은 2장에서 데이터베이스 커넥션 풀과 관련된 기존 연구에 대해서 살펴보고, 3장에서는 실제 검색 클라이언트 객체 풀의 설계에 대해서 설명한다. 이어 4장에서는 3장에서 제안한 검색 클라이언트 풀을 실제로 구현하여 웹 검색 시스템에 적용한 실험결과에 대해서 설명하고, 마지막으로 5장에서는 결론 및 향후 계획에 관하여 기술한다.

2. 관련연구

동적인 HTML 페이지를 사용자에게 제공하는 웹 사이트의 경우 데이터베이스에 저장된 정보를 사용해서 동적인 정보를 만들어낸다. 한 페이지에 대한 사용자의 요청을 처리하기 위해서는 데이터베이스에 대한 하나의 연결이 필요하다. 그러나 데이터베이스 서버에 접속하고 로그인 하는 데는 많은 시간과 자원이 소모된다. 따라서 데이터베이스에 대한 하나의 연결을 만들고 이 연결을 계속적으로 재사용해서 여러 사용자의 요청을 처리하게 되면 상당한 성능 향상을 얻을 수 있다.[4] 현재 데이터베이스와 연동하는 웹 어플리케이션의 경우 데이터베이스 연결 풀을 사용하는 것이 일반적이며, Jakarta DBCP[8], SQL Relay등과 같은 여러 제품들이 사용되고 있다.

3. 검색 클라이언트 객체 풀 설계

검색 클라이언트 객체를 재사용 과정은 도서관에서 책을 빌려보는 과정과 비슷하게 생각해 볼 수 있다.[5] 책을 보고 싶을 때 직접 책을 사는 것보다는 도서관에 가서 책을 빌리는 쪽이 훨씬 더 저렴하다. 이와 비슷하게 각각의 사용자 요청에 대해서 사용자 단독의 검색 클라이언트 객체를 만들어서 제공해 주는 것 보다는 공용의 검색 클라이언트 객체 풀을 만들어 두고, 이를 공유하는 쪽이 메모리 사용의 측면이나 속도 면에서 더 이익이다. 도서관에서 사용자가 책을 빌려보는 과정은 책을 빌리는 과정(Check Out)과 빌린 책을 반납하는 과정(Check In)으로 이루어져 있다. 도서관과 마찬가지로 우리가 설계할 검색 클라이언트 객체 풀도 사용자의 요청에 대해서 검색 클라이언트를 제공하는 과정과 사용이 끝난 검색 클라이언트 객체를 반납 받는 과정으로 이루어져 있다. 한 가지 차이점은 도서관은 장소의 한정 때문에 책을 무한대로 보관할 수 없으므로 특정 책에 대해서는 대출자의 요청을 있을 경우라도 이를 처리해 줄 수 없는 경우가 있을 수 있다. 하지만 검색 클라이언트 객체 풀의 경우에는 객체 풀에 가용한 검색 클라이언트 객체가 없는 경우에는 새로 객체를 만들어 제공해 줄 수 있다.

3.1 사용자의 요청에 대해서 검색 클라이언트 객체를 제공하는 과정(Check Out)

검색 클라이언트 객체 풀은 두 개의 HashTable로 구성된다. 두 개의 HashTable은 현재 사용 중인 검색 클라이언트 객체로 이루어진 used HashTable과 현재 이용 되고 있지 않은 검색 클라이언트 객체로 이루어진 free HashTable로 구성된다. 검색 클라이언트 객체를 얻기 위한 사용자의 요청이 들어오면 일단 free HashTable을 살

펴본다. 사용 가능한 검색 클라이언트 객체가 있으면 이를 사용자에게 제공하고 free HashTable에서 이 객체를 빼서 used HashTable에 넣는다. 사용 가능한 검색 클라이언트 객체가 하나도 없으면 새로운 검색 클라이언트 객체를 생성하고 이를 사용자에게 제공한 후, 마찬가지로 used HashTable에 넣는다.

일반적인 객체 풀에서는 생성 가능한 객체의 개수를 제한하는 경우도 있지만, 이 논문에서는 메모리가 가용한 경우에는 생성 객체의 수를 제한하지 않았다. 즉 계속적으로 사용자 요청이 증가하는 경우에도 메모리가 충분하다면 지속적으로 검색 클라이언트 객체를 만들어서 사용자에게 제공하도록 하였다.

3.2 사용이 끝난 검색 클라이언트 객체를 반납 받는 과정(Check In)

검색 서버에 접속해서 검색 결과를 가져온 후, 이를 사용자에게 보여주고 난 뒤의 검색 클라이언트 객체는 다시 검색 클라이언트 객체 풀로 반납된다. 이는 빌린 책을 다 읽은 대출자가 도서관에 책을 반납하는 과정과 동일하다. 사용자에게 검색 결과를 보여주고 나면 CheckIn 메소드를 사용해서 사용이 끝난 검색 클라이언트 객체를 반납 처리한다. CheckIn 메소드는 해당 검색 클라이언트 객체를 used HashTable에서 free HashTable로 옮겨주는 과정을 처리한다.

일반적인 객체 풀에서는 객체가 처음 만들어 질 때에 생성된 시간을 기록해서 일정 시간이후에는 객체를 없애버리기도 하나, 이 논문에서는 적어도 일정한 사용자 숫자가 계속해서 유지된다고 가정하고 객체를 없애는 작업이 하지 않았다.

4. 실험결과

이 장에서는 3장에서 제안한 검색 클라이언트 객체 풀의 설계를 구현하여 기존의 웹 검색 시스템에 적용한 후, 객체 풀을 사용하였을 때와 사용하지 않을 때의 성능 차이를 웹 성능 측정 도구를 사용해서 비교한 실험 결과를 보여준다. 알고리즘은 Java 언어로 구현되었으며, 웹 어플리케이션 서버로는 Resin 3.0.13을 사용하였다. 전체 웹 시스템의 성능 측정은 Intel Pentium 4HT, 3000MHz CPU와 1G RAM을 가진 시스템에서 Apache JMeter[6]를 사용하여 측정하였다. 가상 사용자의 숫자를 10에서부터 20, 50, 100명으로 점진적으로 증가시켜가면서 성능 측정을 하였는데, 한 대의 검색 서버에서 초당 10건 이상의 검색을 동시에 처리하기 힘들고 또한 일단 검색 클라이언트 객체가 만들어진 후에는 검색 서버로 검색 요청을 보내고 검색 결과를 받아서 처리해 주는 부분은 객체 풀의 사용여부와 상관없이 동일하기 때문에 검색 클라이언트 객체가 만들어지는 부분까지만 구현하여 성능을 측정 비교하였다.

4.1 성능 측정 결과

웹 시스템의 성능 측정은 크게 각각의 사용자의 요청을 처리하는 데 걸린 평균 응답 시간(Average Response Time)과 단위 시간 당 시스템이 처리한 최대 사용자의 숫자(Throughput)로 나누어진다. 본 논문에서는 일정 시간(10~20분)동안 사용자의 검색어를 처리하는 페이지에 HTTP 요청을 보내서 해당 요청이 처리 될 때까지의 시간을 측정하는 방식으로 실험을 진행하였다. 다음 표에서 첫 번째 열은 실험 시간동안 처리된 HTTP 요청의 개수를 나타내며, 두 번째 열은 한 HTTP 요청을 처리하는 데 걸린 평균 시간을 나타내며 단위 시간은 msec이다. 세 번째 열은 한 HTTP 요청을 처리하는 데 걸린 최소 시간을, 네 번째 열은 최대 시간을 나타낸다. 마찬가지로 단위 시간은 msec이다. 다섯 번째 열은 HTTP 요청을 처리하는 데 발생한 에러 비율을 나타내며, 마지막 열은 실험 시간동안 처리된 전체 HTTP 요청의 개수를 실험 시간으로 나눈 단위 시간(1초)당 처리된 HTTP 요청의 숫자를 나타낸다. 즉 표 1에서처럼 초당 10명의 가상 사용자가 시스템에 검색 요청을 보내는 경우에 시스템은 객체 풀링을 적용하기 전에는 초당 최대 453.2 개의 요청을 처리할 수 있었고, 객체 풀링을 적용하게 되면 초당 최대 841.8 개의 요청을 처리할 수 있다는 것이다. 따라서 객체 풀링을 적용한 경우 전체 시스템의 처리 능력이 85% 정도

늘어났음을 알 수 있다. 또한 각각의 사용자에 대한 평균 응답 시간 또한 47% 정도 빨라졌음을 알 수 있다.

	Count	Average	Min	Max	Error%	Rate
객체 풀링 전	272,879	21	0	656	0%	453.2/sec
객체 풀링 후	515,992	11	0	453	0%	841.8/sec

<표 1> 가상 사용자가 10명일 경우

	Count	Average	Min	Max	Error%	Rate
객체 풀링 전	246,934	46	0	985	0%	429.4/sec
객체 풀링 후	484,423	26	0	796	0%	737.7/sec

<표 2> 가상 사용자가 20명일 경우

가상 사용자가 20명일 경우 전체 시스템의 처리 능력은 72% 정도 늘어났으며, 평균 응답 시간은 43% 정도 빨라졌다.

	Count	Average	Min	Max	Error%	Rate
객체 풀링 전	248,390	113	0	6,546	0%	440.7/sec
객체 풀링 후	435,160	68	0	4,594	0%	728.0/sec

<표 3> 가상 사용자가 50명일 경우

가상 사용자가 50명일 경우 전체 시스템의 처리 능력은 65% 정도 늘어났으며, 평균 응답 시간은 39% 정도 빨라졌다.

	Count	Average	Min	Max	Error%	Rate
객체 풀링 전	537,036	207	0	5,828	0%	482.0/sec
객체 풀링 후	800,253	127	0	5,843	0%	784.3/sec

<표 4> 가상 사용자가 100명일 경우

가상 사용자가 100명일 경우 전체 시스템의 처리 능력은 62% 정도 늘어났으며, 평균 응답 시간은 38% 정도 빨라졌다.

가상 사용자의 숫자가 증가하면서 즉, 시스템에 걸리는 부하가 높아질수록 시스템 처리 능력의 향상과 평균 응답 시간의 향상이 줄어들기는 하지만, 객체 풀링을 적용하였을 경우와 객체 풀링을 적용하지 않았을 경우의 성능 차이가 있음을 알 수 있다.

5. 결론 및 향후 연구과제

이상과 같이 웹 검색 시스템에 사용되는 검색 클라이언트에 대해서 객체 풀링 기법을 적용하여 시스템의 성능 차이를 비교 측정해 보았다. 실험 결과에서 알 수 있듯이 데이터베이스 커넥션 풀(Database Connection Pool)의 경우와 마찬가지로 검색 클라이언트 객체 풀의 경우에도 단위 시간당 최대 처리건수(Transaction Per Second) 및 평균 응답 시간에서 상당한 성능 향상을 얻을 수 있음을 알 수 있다.

향후 연구 과제로 본 논문에서 제안한 검색 클라이언트 객체 풀을 여러 검색 서버를 가진 실제 웹 검색 시스템에 적용한 후, 시스템 성능을 비교 측정하는 연구가 필요하다.

[참고문헌]

[1] 최범균, "Jakarta Project", 465p, 가메출판사, 2004
[2] 이원영, 서블렛 + JDBC 연동시 코딩 고려사항 -제2탄-, http://javaservice.net/
[3] Stacy Joines, Ruth Willenborg, Ken Hygh, "Performance Analysis for Java Web Sites", 464p, Addison Wesley, 2002
[4] Hans Bergsten, Improved Performance with a Connection Pool,
http://www.webdevelopersjournal.com/columns/connection_pool.html
[5] Thomas E. Davis, Build your own ObjectPool in Java to boost app speed,
http://www.javaworld.com/javaworld/jw-06-1998/jw-06-object-pool.html
[6] Apache JMeter. http://jakarta.apache.org/jmeter/
[7] KRISTAL-2000, http://giis.kisti.re.kr/
[8] Open Source Database Connection Pools, http://java-source.net/open-source/connection-pools