분산시각 미디어 검색 프레임워크의 성능향상을 위한 부하분산 시스템

심준용° 원재훈 김세창 김정선
한양대학교
{jyshim°, jhwon, sckim, jskim}@cse.hanyang.ac.kr

A load Balancing System for improving the Performance of Semantic Web based Visual Media Retrieval Framework

Junyong Shim°, Jaehoon Won, Sehchang Kim, Jungsun Kim
Dept. of Computer Science and Engineering, Hanyang University

요 약

기존의 Ontology를 이용한 이미지 검색 시스템이나 간단한 구조를 가진 메타데이터 기반의 분산 이미지 검색 시스템들의 단점들을 극복하기 위해 다양한 이미지 제공자들의 자율성을 보장하면서, Semantic 기반의 이미지 검색을 지원하는 분산 시각미디어 검색 프레임워크인 HERMES(The Retrieval Framework for Visual Media Service)가 제안되었다. 분산 환경에서는 시스템의 규모가 커지면서 사용자들의 상호작용 성능을 떨어뜨리지 않으면서 다수의 동시 사용자들을 처리할 수 있는 확장성(Scalability)이 중요한 이슈가 된다. 제안된 프레임워크에서는 서비스를 사용하는 다수의 사용자들이 Broker 서버에 동시에 접속했을 경우 발생하는 Overhead에 대한 문제를 해결 할 수 없었기 때문에 성능의 저하와 확장성을 고려할 수 없는 문제를 안고 있다. 이런 문제를 해결하기 위해서 Broker 서버의 내부 컴포넌트의 수행시간을 측정하고 이를 주기적으로 수집하여 저장하는 Monitoring System이 추가로 연구되었지만, 수집한 정보를 가공하여 다수의 Broker 서버에 대한 부하를 분산하는 알고리즘은 제공되지 않았다. 본 논문에서는 다수의 동시 사용자들이 접속했을 경우에도 성능의 저하 없이 비슷한 수준의 서비스를 제공하기 위해서 Broker 서버를 증설하여 Monitoring System으로부터 각각의 Broker 내부 컴포넌트의 수행시간을 측정하여 저장하고, 저장된 데이터에 대하여 각 Broker들에 대한 우선순위를 결정하는 테이블을 작성한다. 사용자로부터 Query를 입력받는 User Interface는 Broker의 Ranking Table을 참조하여 다수의 Query 수행을 여러 서버로 분산처리하게 함으로써 성능에 대한 신뢰성을 향상 시킬 수 있는 Load Balancing System을 제안한다.

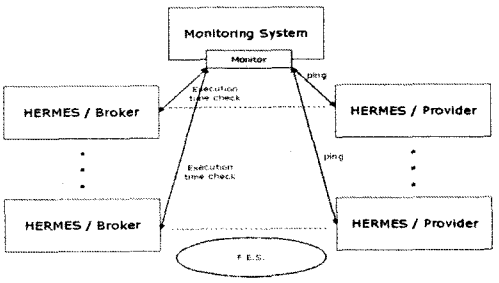
1. 서론

기존의 Ontology를 이용한 이미지 검색 시스템이나, 간단한 구조를 가진 메타데이터 기반의 분산 이미지 검색 시스템들의 단점들을 극복하기 위해 다양한 이미지 제공자들의 자율성을 보장하면서, Semantic 기반의 이미지 검색을 지원하는 프레임워크인 HERMES(The Retrieval Framework for Visual Media Service)가 연구되었다[1][2].

HERMES Architecture는 HERMES/B, HERMES/P, F·E·S Provider와 같은 세 개의 주요 컴포넌트들로 구성 되어있다. 주요 특징을 살펴보면 각 컴포넌트들 간의 플랫폼 독립적인 Web Services를 이용하여 이기종간의 서비스를 가능하게 하였고, 메타데이터와 Ontology를 이용한 지능적인 Visual Media data 검색을 지원한다.

분산 환경에서는 시스템의 규모가 커짐에 따라 사용자들의 상호작용 성능을 떨어뜨리지 않으면서 다수의 동시 사용자들을 처리할 수 있는 확장성(Scalability)이 이슈가 된다[3]. 제안된 프레임워크에서는 다수의 사용자들의 요구에 인한 Broker 서버의 Overhead를 줄이기 위하여 측정 데이터가 제공되지 않았기 때문에 Broker서버 내부의 컴포넌트 수행시간(Broker 서버의 컴포넌트를 수행하고 Provider를 호출하여 원하는 정보를 사용자에게 제공할 때까지의 시간)을 측정하여 저장하는 Monitoring System이 연구되었다[4]. Monitoring System의 Architecture는 아래 [그림 1]과 같다.

하지만 Monitoring System에서는 Broker 서버의 수행시간만 측정하여 저장할 뿐 수집된 정보를 가공하여 Broker의 부하를 분산하기 위한 알고리즘은 제공되지 않았다.



[그림 1] Monitoring System Architecture

본 논문에서는 다수의 동시 사용자들의 접속에 의한 Broker 서버의 Overhead에 대한 문제를 해결하여 성능과 확장성을 높일 수 있는 Load Balancing System을 제안한다.

이 시스템은 다수의 Broker 서버 운영을 전제로 하며 Monitoring System으로부터 각각의 Broker 서버의 내부 컴포넌트의 수행시간을 측정하여 저장한다. 또한 컴포넌트의 수행시간은 분산 환경에서의 네트워크 트래픽이 전체적인 수행능력에서 큰 비중을 차지하므로 Broker 서버에서 Provider 서버들을 호출하여 사용자에게 응답이 되는 시간까지 포함하여 측정한다. 저장된 각 Broker들의 컴포넌트 수행시간에 대한 우선순위를 결정하고 테이블을 작성함으로써 다수의 사용자의 Query를 수행할 때 Broker 서버들에 대한 Ranking Table을 참조하여 여러 서버로 분산처리하게 하여 성능에 대한 신뢰성을 향상 시킨다.

2. 관련 연구

2.1 Load Balancing Metrics

Load Balancing은 다수의 사용자들이 서버에 서비스를 요청할 때 Overhead에 대하여 서버의 병목현상을 해결하고 자원에 대한 부하를 분산하는 방법이다. 이러한 부하 분산을 위하여 시스템으로부터 필요한 Metric은 다음과 같다[5].

2.1.1 CPU share

컴포넌트가 호출됨으로써 생기는 CPU 점유율의 변화량

2.1.2 Communication Delay

서버-클라이언트 환경에서 사용자가 서버에게 서비스 요청 시 응답을 받기까지의 네트워크 지연시간

2.1.3 Host Idleness

Process를 수행하지 않거나 오랫동안 휴면 상태에 있는 Host

2.1.4 Component Execution Time

서비스를 제공하는 컴포넌트들이 시작해서 끝날 때까지의 각 Method의 수행시간

HERMES는 Broker 서버의 Overhead뿐만 아니라 Provider 서버로의 Web Services 호출에 의한 응답시간이 사용자들에게 전체적인 Overhead로 작용한다. 따라서 Broker의 Component Execution Time뿐만 아니라 Broker 서버와 Provider 서버들 간의 Communication Delay를 모두 고려하여 Load Balancing Scheduling에 필요한 Metric으로 사용한다.

2.2 분산 환경의 Load Balancing Policy

분산 환경에서 Load Balancing Policy는 작업 전이(Job transfer)의 결정이 서버의 현재 상태를 즉각적으로 반영하는지 여부에 따라 Dynamic과 Static 두 가지로 나뉜다[6].

2.2.1 Dynamic

부하 분산을 위한 서버를 결정할 때 현재의 상태나 최근의 상태를 반영하므로 시스템 상태에 대한 정보를 요청할 때 정보 수집에 대한 Cost가 필요하지만 분산 환경에서 동적으로 변화하는 서버의 상태를 반영할 수 있다.

2.2.2 Static

시스템 상태에 대한 정보가 이미 서비스 전에 결정이 되어져 있기 때문에 정보를 수집할 필요가 없게 된다. 따라서 알고리즘이 단순하고 Cost가 적게 든다. 하지만 시스템의 상태가 자주 변화하는 분산 환경에서는 변화에 대한 Update가 늦기 때문에 제한된 성능을 제공하거나 불균형적인 Load Balancing이 이루어 질수 있다.

HERMES는 Dynamic 방법을 기초로 기존에 제안되었던 Monitoring System을 통하여 실시간으로 Broker 서버의 컴포넌트 수행시간과 Provider 서버의 호출에 의한 응답시간을 저장하여 사용하지만 관리자가 정해놓은 주기를 기준으로 Broker에 대한 테이블을 만들어서 Load Balancing을 수행한다.

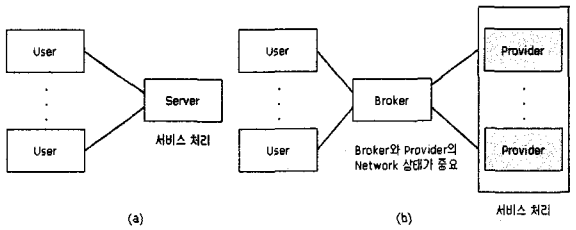
2.3 Load Balancing을 위한 Scheduling

분산 환경에서의 서버의 병목현상은 네트워크의 트래픽에 많은 영향을 주게 된다. 이러한 네트워크의 트래픽을 고려한 Load Balancing이 필요하다. 기존의 부하를 분산하기 위해 가상 서버에서 사용하는 Scheduling 알고리즘에는 라운드 로빈(Round-Robin), 가중치 기반 라운드 로빈(Weighted Round-Robin), 최소 접속(Least-Connection), 가중치 기반 최소 접속(Weighted Least-Connection) Scheduling 방법이 있다[7].

라운드 로빈 알고리즘은 서버 연결의 개수나 네트워크의 응답시간 등은 고려하지 않고 서버를 순서대로 Scheduling 하는 방식이다. 이 방식은 네트워크의 상태를 고려하지 않기 때문에 실제 서버사이에 동적인 부하 불균형이 심각해질 수 있다. 최소 접속 알고리즘은 서비스를 사용하는 사용자의 접속이 가장 적은 서버로 요청을 직접 연결하는 방식으로 각 서버에서 동적으로 실제 접속한 사용자의 연결개수를 파악하여 Scheduling 하는 알고리즘이다. 가중치 기반 라운드 로빈 알고리즘과 가중치 기반 최소 접속 알고리즘은 위에서 설명한 각각의 Scheduling과 같은 방식으로 동작하지만 각 서버에 가중치(관리자가 지정한 정수 값)를 부여하여 서버마다 서로 다른 처리 용량을 정하는 방식이다.

각각의 방식들은 분산 환경에서의 Load Balancing을 위한 Scheduling 기법들이지만 단순히 분산되어 있는 서버를 정해진 순서대로 Scheduling하거나 서버에 접속한 사용자의 수만을 가지고 Scheduling 하기 때문에 각 서버들이 가지고 있는 네트워크와 네트워크 간의 트래픽 상태는 반영할 수 없다.

HERMES는 Broker 서버가 사용자의 요청을 직접 처리하지 않고 Provider 서버들에게 Web Services 요청을 하기 때문에 Broker 서버의 작업순서나 접속자 수만을 가지고 Scheduling 하기에는 균형적인 Scheduling을 할 수가 없다. 따라서 Provider 서버들의 네트워크 트래픽 상태를 고려하여 Scheduling 하는 방식을 사용한다.[그림 2]

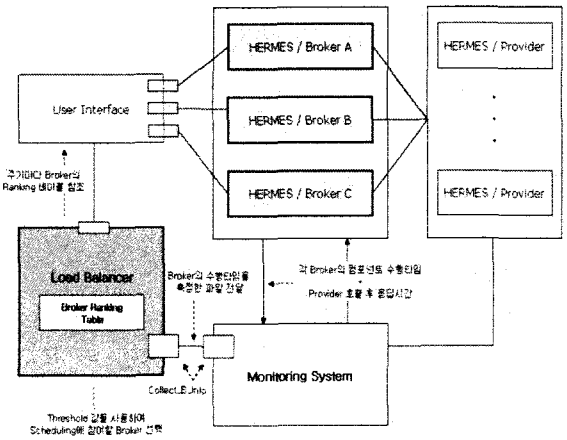


[그림 2] 일반적인 서버(a)와 HERMES 서버(b)의 비교

3. Architecture

본 논문에서는 Load Balancing Scheduling을 할 때 Broker 서버와 Provider 서버 사이의 네트워크 트래픽을 고려할 수 있도록 Broker 서버의 컴포넌트 수행시간과 Broker 서버에서 Provider 서버로 Web Services 호출 후 사용자가 요청한 Visual Medial Data를 검색하여 응답을 할 때까지의 네트워크 트래픽 타임을 모두 측정한다.

제안하는 Load Balancing System Architecture는 [그림 3]과 같다.



[그림 3] Load Balancing System Architecture

3.1 Load Balancing System의 구성요소

Monitoring System은 각 Broker 서버의 컴포넌트 수행시간과 Provider를 호출하여 사용자가 원하는 정보에 대하여 응답을 받을 때까지의 네트워크 트래픽 타임을 각각의 Broker 서버마다 관련된 파일을 만들어서 저장한다. 저장된 파일은 Load Balancer로부터 호출되어 Broker 서버의 Ranking Table을 만들기 위한 데이터로 사용된다. User Interface는 Load Balancer로부터 만들어진 Ranking Table을 참조하여 사용자들의 Query를 Scheduling한다.

Load Balancing System에서 사용되는 주요 컴포넌트의 기능들은 다음과 같다.

3.1.1 Collect_B_Info컴포넌트

Monitoring System은 각 Broker 서버의 컴포넌트 수행시간과 Broker와 Provider 서버간의 네트워크 트래픽 타임을 파일로 저장한다. Load Balancer의 Collection 컴포넌트는 관리자가 정해놓은 주기마다 Monitoring System으로부터 저장된 파일을 전송받는다.

3.1.2 Ranking Table 컴포넌트

Ranking Table 컴포넌트는 Monitoring System으로부터 받은 각 Broker에 대한 파일을 열어서 전체 수행시간을 Query의 개수로 나눈 평균 수행시간을 계산한 후 각 Broker의 평균 수행시간을 모두 더하여 Broker 서버의 개수로 나눈 값을 구한다.

제안된 논문에서는 이 값을 Broker 서버를 부하 분산 Scheduling에 포함시키기 위한 Threshold 값으로 사용한다. 이 내용을 요약하면 다음과 같다.

- Broker 서버의 평균 수행시간
  =Broker의 컴포넌트 수행시간의 합 / 수행된 Query의 개수
- Threshold
  =모든 Broker의 평균 수행시간의 합 / Broker 서버의 개수

Threshold 값과 Broker 서버의 평균 수행시간을 비교하여 Broker 서버의 값이 작으면 Load Balancing Scheduling에 포함되고, 그렇지 않으면 제외 된다. 다음 주기에서도 Scheduling을 하기 위한 Broker 서버를 선택할 때에는 동일한 방법으로 수행되며 이전 주기에 제외되었던 Broker 서버는 무조건 Scheduling에 참여시킨다.

3.1.3 컴포넌트들 간의 협력관계

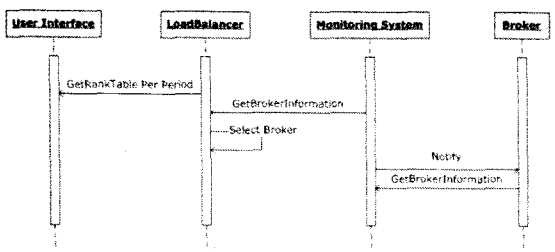
- User Interface

사용자의 Query를 처리하기 전 Load Balancer에게 Broker 서버의 List를 받아오기 위해 Web Services로 통신

- Load Balancer

Monitoring System과 Remote Object를 사용하여 Broker 서버의 저장된 데이터 파일들을 가져오기 위해 통신, 데이터 파일들을 열어서 Broker 서버의 우선순위를 정하여 User Interface와 Web Services로 통신한다.

Load Balancer와 연결되어 있는 컴포넌트들과의 협력관계를 다이어그램으로 나타내면 [그림 4]와 같다.

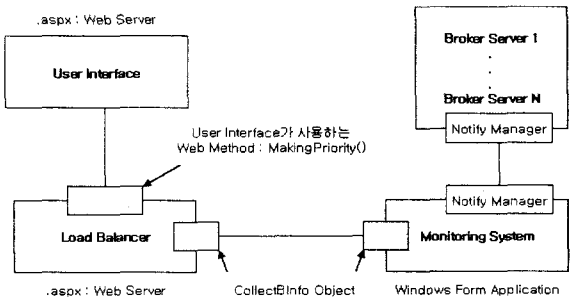


[그림 4] 컴포넌트들 간의 Sequence Diagram

3.2 Load Balancer 구현기술

Load Balancer는 Monitoring System으로부터 Broker 서버의 정보를 받기 위해서 Remote Object인 CollectBInfo 객체를 포함한다. 이 Remote Object를 만들기 위해서 .Net Remoting을 사용했다. 또한 Monitoring System으로부터 받은 각 Broker 서버의 수행시간들을 가지고 우선순위를 결정한 후 주기적으로 User Interface에게 Broker의 List 테이블을 전달해 줄 수 있는 GetBrokerList() Method는 Web Services를 사용했다.

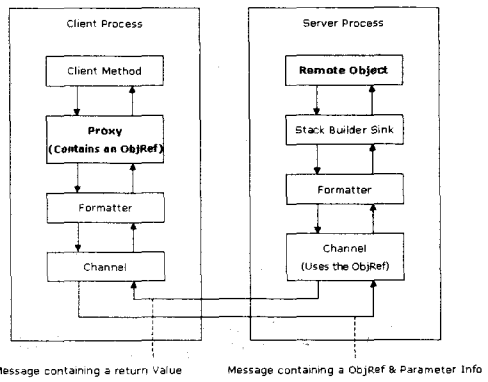
Load Balancer가 가지고 있는 Remote Object와 Web Method를 포함한 구조를 보면 [그림 5]와 같다.



[그림 5] Load Balancer의 Remote Object와 Web Method

3.2.1 Broker 서버의 정보 수집을 위한 .Net Remoting

- .Net Remoting의 Architecture는 [그림 6]과 같다[8][9].



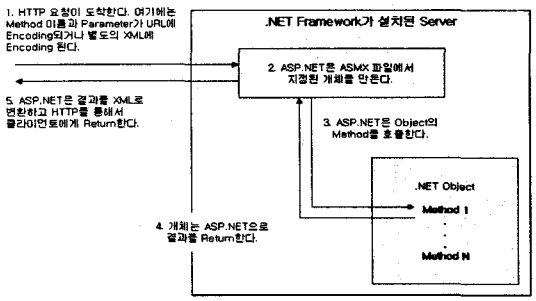
[그림 6] .Net Remoting Architecture

- 개념 및 구성요소

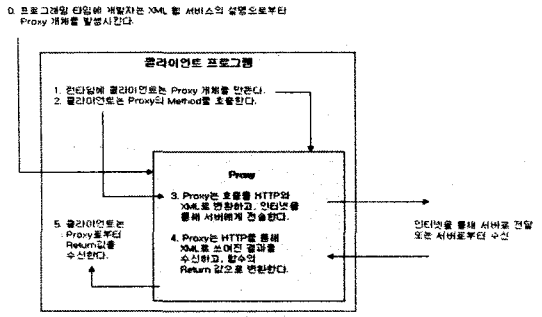
클라이언트 프로그램이 원격 객체의 Method를 지역 객체의 Method처럼 호출 할 수 있게 해주는 기술로써 Remote Object, Channel(HTTP, TCP), Formatter(SOAP, TCP)의 세 가지 요소로 구성되어 있다. [그림 5]의 CollectBInfo Object는 Broker 서버의 정보를 제공해주는 Method를 포함하고 있는 Remote Object이며, Channel은 HTTP Channel과 Formatter는 Soap Formatter를 사용하였다.

3.2.2 Broker List 참조를 위한 Web Method

- Web Services의 서버 측과 클라이언트 측 관점에서 Architecture를 살펴보면 [그림 7], [그림 8]과 같다[10].



[그림 7] Web Services의 서버 측 Architecture

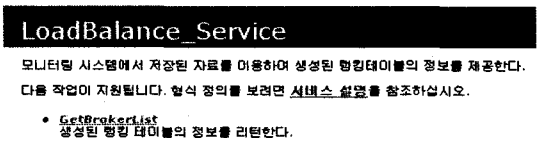


[그림 8] Web Services의 클라이언트 측 Architecture

• 개념 및 구성요소

클라이언트 측에서는 XML 웹 서비스의 설명을 읽어서 Proxy를 만들고, 클라이언트를 개발하기 위해 사용한 프로그래밍 언어로 작성된 함수를 넣어둔다. 클라이언트는 이 함수 중 하나를 호출할 때, Proxy 클래스는 HTTP 요청을 만들어서 서버로 보낸다. 서버 측 ASP.NET은 HTTP를 통해 전송되는 클라이언트 요청을 받아들이고 이를 Object의 호출에 매핑 시킨다.

Load Balancer가 User Interface에게 Broker Ranking Table의 List를 전달해주는 Web Method의 인터페이스를 웹 브라우저로 확인하면 [그림 9]와 같다.

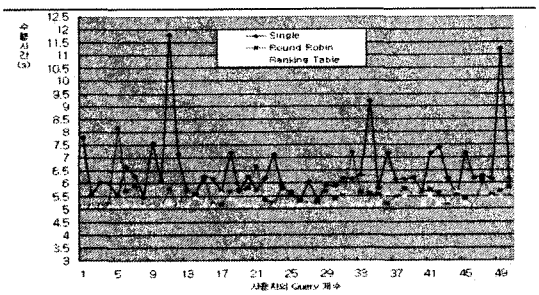


[그림 9] Load Balancer의 UI와 협력하는 Web Method

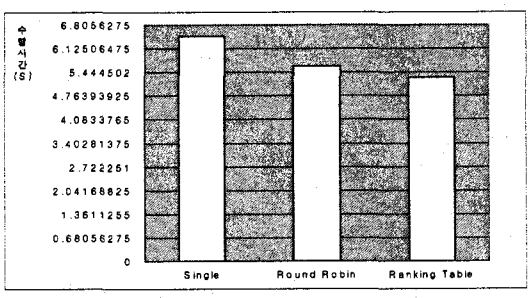
4. 성능 분석

실험 환경은 Broker 서버 3대와 Provider 서버 10대를 설치하여 Load Balancing을 적용하지 않은 Broker 서버 1대, Round-Robin Scheduling 기법을 적용한 Broker 서버 3대 마지막으로 논문에서 제안한 Ranking Table 기반 Scheduling 기법을 적용한 Broker 서버 3대를 각각 구축하여 임의의 50개의 Query를 수행시켜 응답되는 시간을 그래프로 나타내고 비교 분석하였다. 또한 모니터링 시스템을 통하여 Query가 입력될 때 Broker 서버가 Ranking Table에 맞게 Scheduling 되는지를 확인해 보았다.

Query의 수행시간을 비교한 그래프와 평균 수행시간을 비교한 그래프를 살펴보면 Load Balancing을 적용한 Broker 서버가 전체적으로 Component의 수행시간 면에서 좋은 성능을 보였고, Ranking Table Scheduling 방법이 Round-Robin Scheduling 방법보다 좋은 성능을 보이는 것을 [그림 10]과 [그림 11]에서 확인할 수 있다.

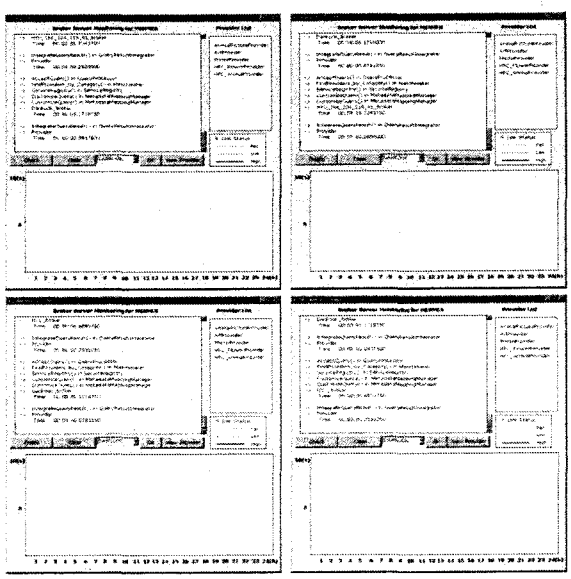


[그림 10] Query 50개에 대한 각각의 수행시간 그래프



[그림 11] Query 50개에 대한 평균 수행시간 그래프

Monitoring System을 통하여 Broker 서버가 Ranking Table에 의하여 Scheduling 되는지를 [그림 12]에서 확인할 수 있다.



[그림 12] Broker 서버의 Scheduling 스냅 샷

5. 결론 및 향후 과제

본 논문은 기존의 HERMES의 Broker 서버에 다수의 동시 접속자들이 Query를 수행할 경우 발생할 수 있는 Overhead를 줄이기 위해서 Broker 서버를 증설하고 Load Balancing 하기 위한 시스템을 구축하였다. 또한 Broker 서버의 내부 컴포넌트 수행시간과 Broker 서버와 Provider 서버간의 Web Services 호출에 의한 네트워크 트래픽 시간을 함께 측정하여 Load Balancing 할 경우 Broker 서버의 우선순위를 결정할 수 있는 Scheduling 방법을 연구하였다.

실험을 통하여 분산 환경에서의 대표적인 Load Balancing Scheduling 방법 중 하나인 Round-Robin Scheduling과 본 논문에서 제안한 방법을 비교한 결과 제안한 방법이 평균적으로 더 좋은 성능을 보이는 것을 실험에 대한 결과 그래프를 통하여 확인하였다. 또한 분산 시스템에서는 서비스를 직접 처리하는 서버의 상태가 Load Balancing을 위한 중요한 Metric이 되었지만 HERMES의 Broker 서버의 경우 서비스를 처리하는 Provider 서버와의 네트워크 상태 또한 Broker 서버의 상태만큼 중요하다는 것을 Round-Robin 방법과 비교한 결과 알 수 있었다.

향후 과제로서 현재의 HERMES Broker 서버는 내부 컴포넌트에 대한 멀티 쓰레딩이 지원되지 않고 있기 때문에 다수의 Query가 Provider 서버로의 Web Services 호출이 발생하기 전까지는 다음 Query들이 작업 큐에서 대기해야된다. 물론 Broker 서버는 사용자가 원하는 서비스를 직접 처리 하지 않기 때문에 일정양의 Query에 대해선 문제가 없겠지만 다수의 접속자와 시스템이 확장될 수 있다는 것을 고려하면 Overhead에 대한 수행능력의 저하는 피할 수 없다. 또한 Provider 서버에 대한 Web Services 호출이 잦으므로 UDDI를 사용한 Web Services의 QoS 평가가 연구된다면 성능과 신뢰성 면에서 더 좋아질 수 있다.

6. 참고 문헌

[1] 양명미, 정병훈, 손영수, 김정선, "Ontology를 이용한 Web Services 기반 분산 이미지 검색 프레임워크 Architecture", 한국컴퓨터종합학술대회 2005 논문집 Vol.32, No.1(B)

[2] 나연욱, 이복주, 김정선, "Visual Media Retrieval Framework Using Web Service", Lecture Notes in Computer Science 3597 권 pages 104-113, Jul, 2005

[3] M. Macedonia and M. Zyda, *Networked Virtual Environments Design and Implementation*, Addison Wesley, 1999

[4] 심준용, 김세창, 원재훈, 김정선, "분산 시각미디어 검색 프레임워크를 위한 모니터링 시스템", 2006 한국컴퓨터종합학술대회 논문집 Vol. 33, No. 1(B)

[5] Kristian Paul Bubendorfer, "Resource Based Policies for Load Distribution", Victoria University of Wellington, Aug, 1996

[6] T.V. Gopal, N.S. Karthic Natarak, C. Ramaurthy and V. Sankaranarayanan, "Load Balancing in Heterogenous Distributed Systems", Microelectron. Reliab., Vol.36, No.9, pp. 1279-1286, 1996.

[7] "리눅스 가상 서버 프로젝트", http://linuxvirtualserver.org

[8] David Conger, "Remoting with C# and .NET", Wiley Publishing, Inc., 2003, ISBN 0-471-27352-X

[9] Microsoft's .net Remoting: A Technical Overview, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp

[10] Scott Short, "Building XML Web Services For The Microsoft.NET Platform" Microsoft Press., 2002, ISBN 89-5674-034-8