분산시각 미디어 검색 프레임워크의 성능향상을 위한 브로커 서버 우선순위를 이용한 라운드 로빈 스케줄링 기법

(A Scheduling Algorithm using The Priority of Broker for Improving The Performance of Semantic Web-based Visual Media Retrieval Framework)

심 준 용 † 원 재 훈 †† 김 세 창 †† 김 정 선 †††
(Junyong Shim) (Jaehoon Won) (Sechang Kim) (Jungsun Kim)

요 약 기존의 Ontology를 이용한 이미지 검색 시스템이나 메타데이타 기반의 분산 이미지 검색 시스템들의 단점들을 극복하기 위해 이미지 제공자들의 자율성을 보장하면서, Semantic 기반의 이미지 검색을 지원하는 분산 시각미디어 검색 프레임워크인 HERMES(The Retrieval Framework for Visual Media Service)가 제안되었다. 제안된 프레임워크에서는 서비스를 사용하는 다수 사용자들이 Broker서버에 동시에 접속했을 경우 발생하는 Overhead에 대한 문제를 해결 할 수 없었기 때문에 성능의 저하와 확장성을 고려할 수 없는 문제를 안고 있다. 본 논문에서는 다수의 동시 사용자들이 접속했을 경우에도 성능의 저하 없이 비슷한 수준의 서비스를 제공하기 위해서 Broker서버를 증설하여 Monitoring System으로부터 각각의 Broker 내부 컴포넌트의 수행시간을 측정하여 저장하고, 저장된 데이타에 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 향상시킬 수 있는 Load Balancing 시스템을 제안한다. 또한 기존의 방식들과 비교실험을 통하여 제안하는 Load Balancing 시스템의 스케줄링 기법이 빠르다는 것을 보여준다.

키워드 : 웹서비스, 온톨로지, 시각 미디어 검색 프레임워크, 부하분산, 모니터링

Abstract To overcome the weakness of the image retrieval system using the existing Ontology and the distributed image based on the database having a simple structure, HERMES was suggested to ensure the self-control of various image suppliers and support the image retrieval based on semantic. the mentioned framework could not solve the problems which are not considered the deterioration in the capacity and scalability when many users connect to broker server simultaneously. In this paper the tables are written which in the case numerous users connect at the same time to the supply analogous level of services without the deterioration in the capacity installs Broker servers and then measures the performance time of each inner Broker Component through Monitoring System and saved and decides the ranking in saved data. As many Query performances are dispersed into several Servers User inputted from the users Interface with reference to Broker Ranking Table, Load Balancing system improving reliability in capacity is proposed. Through the experiment, the scheduling technique has proved that this schedule is faster than existing techniques.

Key words : Web Service, ontology, the retrieval framework for visual media service, load balancing, monitoring

† 정 회 원 : 한양대학교 컴퓨터공학과
jyshim@lignex1.com
†† 학생회원 : 한양대학교 컴퓨터공학과
jhwon@cse.hanyang.ac.kr
sckim@cse.hanyang.ac.kr
††† 종신회원 : 한양대학교 전자컴퓨터공학부 교수
jskim@cse.hanyang.ac.kr
논문접수 : 2007년 1월 22일
심사완료 : 2007년 12월 11일

정보과학회논문지: 소프트웨어 및 응용 제35권 제1호(2008.1)

1. 서 론

최근 멀티미디어와 유무선 정보통신망의 발전에 따라 이미지나 영상과 같은 시각미디어 검색 서비스에 대한 요구가 다양해지고 있다. 이미지 검색 방법 또한 다양해진 요구를 충족시키기 위해 내용기반 방법(CBIR), 메타데이타 방법(Photo-Book, QBIC)과 같은 Keyword-based 방법부터 Semantic web의 Ontology를 이용한 검색 방법까지 더 지능적이고 정확한 검색 방법이 연구되고 있다. 하지만 기존의 Ontology를 이용한 이미지 검색 시스템이나, 간단한 구조를 가진 메타데이타 기반의 분산 이미지 검색 시스템들은 특정 도메인에 종속되거나 메타데이타가 표준화되지 못해 이미지 공유와 통합 검색에 문제가 있었다. 이와 같은 기존 연구의 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 시스템은 웹상에 분산 저장된 시각미디어 정보의 Semantic을 보다 잘 표현해서 시각 미디어의 활용도를 높이고 정보 검색을 지능화하려는 연구이다. 시스템의 아키텍처는 HERMES/B, HERMES/P, F·E·S Provider와 같은 세 개의 주요 컴포넌트들로 구성 되어있다. 그림 1은 전체적인 작업의 흐름을 보여주는 아키텍처이다.

주요 특징을 살펴보면 각 컴포넌트들 간의 플랫폼 독립적인 Web Services를 이용하여 이기종간의 서비스를 가능하게 하고, 메타데이타와 Ontology를 이용한 지능적인 시각 미디어 데이타 검색을 지원한다. HERMES/B와 HERMES/P가 서로 Web Services 통신을 하기 때문에 그림 2와 같이 HERMES/P가 서로 다른 스키마를 갖는 데이타를 지원할 수 있다.

HERMES 시스템과 같은 분산 환경 시스템에서는 일반적으로 시스템의 규모가 커짐에 따라 사용자들의 상호작용 성능을 떨어뜨리지 않으면서 다수의 동시 사용자들을 처리할 수 있는 확장성(Scalability)이 이슈가 된

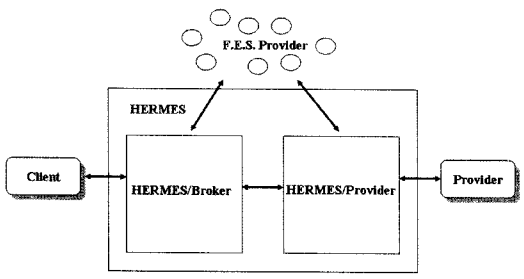


그림 1 HERMES 아키텍처

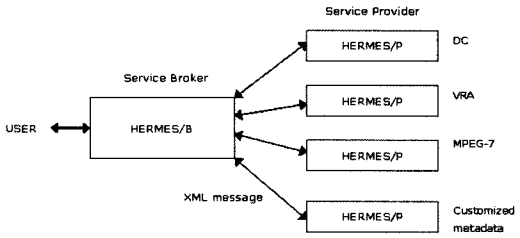


그림 2 HERMES의 주요 컴포넌트

다[3]. HERMES 시스템 또한 Broker와 Provider서버 간의 확장성을 고려해야 한다. 즉, 시각미디어를 제공하는 Provider가 더욱 다양해지고 서비스를 이용하려는 사용자들이 늘어날 수 있다. 하지만 제안된 프레임워크에서는 다수 사용자들의 요구에 의한 Broker서버의 Overhead를 줄이기 위하여 측정 데이타가 제공되지 않았기 때문에 Broker와 Provider서버에 대한 상태를 감시하면서, Broker서버 내부의 컴포넌트 수행시간(Broker서버의 컴포넌트들을 수행하고 선택된 Provider서버를 호출하여 원하는 정보를 사용자에게 제공할 때까지의 전체 시간)을 측정하여 저장할 수 있는 Monitoring System이 연구되었다[4]. Monitoring System의 아키텍처는 아래 그림 3과 같다.

Monitoring System에서는 Broker서버의 수행시간만 측정하여 저장할 뿐 수집된 정보를 가공하여 Broker서버의 부하를 분산하기 위한 알고리즘은 제공되지 않았다.

따라서 본 논문에서는 Monitoring System으로부터 수집된 Broker서버의 정보를 제공받아 다수의 동시 사용자들의 접속 시 설치되어있는 다수의 Broker서버로 분산시킴으로써 성능과 확장성을 높일 수 있는 Load Balancing 시스템을 제안한다. 이 시스템은 부하 분산을 위한 다수의 Broker서버 운영을 전제로 하며 Monitoring System으로부터 각각의 Broker서버의 내부 컴포넌트의 수행시간을 측정하여 저장한다. 또한 컴포넌트의 수행시간외에 분산 환경에서의 네트워크 트래픽이 전체적인 수행능력에서 큰 비중을 차지하는 것을 고려하여 Broker서버로부터 호출된 Provider서버들로부터의 작업처리와 응답시간까지 모두 포함한 Communication Delay도 함께 측정한다. 본 논문에서는 측정된 이 두 가지 값을 합하여 요구-응답시간이라고 칭한다. 저장된 각 Broker서버들의 요구-응답시간에 대한 우선순위를 결정하고 테이블을 작성함으로써 다수 사용자의 Query를 수행할 때 Load Balancing을 위한 Broker서버들의 목록 테이블을 제공하여 Query들을 여러 서버로 분산처리 하게 함으로써 성능에 대한 신뢰성을 향상 시킨다. 일반적으로 분산 시스템에서 부하 분산 스케줄링기법으로 라운드로빈(Round Robin), 가중치 기반 라운드로빈

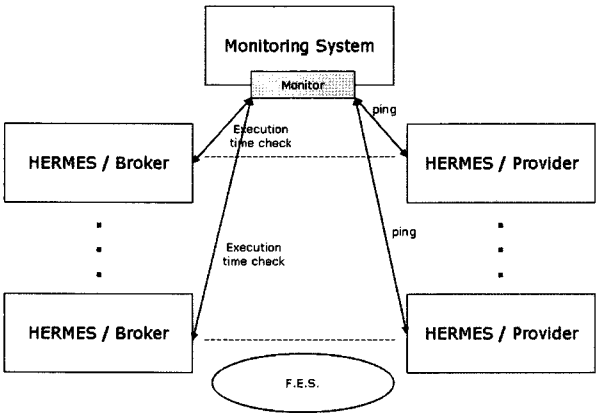


그림 3 Monitoring System 아키텍처

(Weighted Rounded Robin), 최소접속(Least Connection), 가중치 기반 최소접속(Weighted Least Connection)과 같은 방법들이 채택되어 사용되는데 이러한 방법으로는 HERMES 시스템의 특성상 균형 있는 부하 분산 스케줄링을 할 수 없다. 따라서 사용자의 Query를 직접 받아 최적화된 Query를 만들어 적합한 Provider 서버를 선택 해주는 Broker서버와 Broker서버로부터 작업요청을 받아 시각 미디어 데이타를 제공하는 Provider서버 사이의 호출 응답시간이 스케줄링 하는데 중요한 요소가 될 수 있다는 것을 고려하여 전체적으로는 라운드로빈(Round Robin) 방법을 사용하지만 전체 Broker 서버의 요구-응답시간의 평균시간에 미치지 못하는 Broker서버는 한 주기 동안 제외시키는 방법을 적용한다. 또한 Monitoring System으로부터 수집된 Broker서버의 요구-응답시간을 토대로 Broker서버들의 순위를 결정하여 높은 우선순위를 갖는 Broker서버에게 더 많은 Query를 처리하게 함으로써 HERMES 시스템에 적합한 부하 분산 스케줄링 기법을 제시한다.

본 논문의 구성은 다음과 같다. 제2장에서는 일반적인 분산 시스템에서 부하 분산을 위한 기존 연구를 살펴보면서 HERMES 시스템을 위한 적합한 접근 방법을 제시한다. 제3장에서는 본 논문에서 제안하는 Load Balancing 시스템의 아키텍처와 주요 구성 요소의 특징 및 구현 기술을 살펴보고, 제4장에서는 분산 환경에서의 일반적인 스케줄링 기법과 제안하는 스케줄링 기법을 비교 실험하여 제안된 방법의 효율성을 살펴본다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

Load Balancing은 다수 사용자들이 원하는 서버에 서비스를 요청할 때 서버의 병목현상을 해결하고, 자원에 대한 부하를 분산하는 방법으로 서비스를 이용하기 위한 사용자가 증가할수록 서비스를 제공하는 서버의 안정성과 성능유지를 확보하는 분야에서는 중요한 이슈가 된다. 본 장에서는 기존에 연구되었던 Load Balancing을 위한 요소들과 스케줄링 기법들을 살펴보고, 그것과 비교하여 HERMES 시스템에 적합한 측정 요소들과 스케줄링 기법들을 알아본다. 또한 각 Broker서버의 정보를 제공하는 Monitoring System과 Load Balancing 시스템의 통신 방법인 .Net Remoting에 대해서 알아보고, Load Balancing 시스템이 HERMES 시스템의 주요 컴포넌트들과 통신하기 위해 사용하는 Web Services에 대해서 알아본다.

2.1 Load Metric

2.1.1 CPU share

컴포넌트가 호출됨으로써 생기는 CPU 사용률의 변화량으로 컴포넌트를 사용하는 것이 시스템에 어떠한 영향을 주는지 판단하기 위한 요소

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 컴포넌트가 수행될 때 관련된 모든 메서드가 호출되어 반환되는 시간으로 컴포넌트의 전체 성능평가를 위한 요소

HERMES 시스템은 Broker서버의 Overhead뿐만 아니라 Provider서버로의 Web Services 호출에 의한 응답지연 시간 또한 사용자들에게 전체적인 Overhead로 작용한다. 따라서 Broker서버의 Component Execution

Time뿐만 아니라 Broker와 Provider서버들 간의 Communication Delay를 모두 고려한 요구-응답시간을 측정하여 Load Balancing을 위한 Broker서버의 목록 테이블 구성 시 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 아니면 현재의 서버 상태와 상관없이 미리 정해놓은 상태를 반영하는지에 따라 Dynamic 방법과 Static 방법 두 가지로 나뉜다[6].

2.2.1 Dynamic

Load Balancing을 위한 서버 결정을 위하여 Load정보가 필요할 때마다 질의를 통해서 알아내는 방법이다. 동적으로 변화하는 시스템에서는 최신의 서버 상태를 유지할 수 있기 때문에 균형적인 Load Balancing을 유지한다. 하지만 시스템 상태에 대한 정보를 요청할 때마다 네트워크가 좋지 않은 경우 정보 수집에 대한 비용이 커질 수 있다.

2.2.2 Static

Load Balancing을 위한 시스템 상태에 대한 정보가 이미 서비스 전에 결정이 되어져 있기 때문에 정보 수집할 필요가 없게 된다. 따라서 알고리즘이 단순하고 비용이 적게 든다. 하지만 시스템의 상태가 자주 변화하는 분산 환경에서는 변화에 대한 서버의 상태반영이 늦기 때문에 제한된 성능을 제공하거나 불균형적인 Load Balancing이 이루어질 수 있다.

HERMES 시스템은 Dynamic방법을 기초로 하지만 Monitoring System을 통하여 실시간으로 Broker서버의 컴포넌트 수행시간과 Provider서버의 호출에 의한 응답시간을 저장하여 반영하는 시점은 시스템에서 정한 일정한 주기를 기준으로 한다.

2.3 Load Balancing을 위한 스케줄링

분산 환경에서의 서버 병목현상은 서비스를 받고자 하는 사용자들에게 성능에 대한 신뢰성을 떨어뜨린다. 따라서 다수 사용자를 위해서 다수의 서버를 두어 분산처리 하게 함으로써 성능에 대한 신뢰성과 안정성을 유지할 수 있다. 일반적으로 부하를 분산시키기 위해 기존 서버 환경에서 사용하는 스케줄링 알고리즘에는 라운드로빈(Round-Robin), 가중치 기반 라운드로빈(Weighted Round-Robin), 최소 접속(Least-Connection), 가중치 기반 최소 접속(Weighted Least-Connection) 스케줄링 방법이 있다[7].

2.3.1 라운드로빈(Round Robin)

라운드 로빈 방식은 사용자의 요청이 들어오면 서버에 연결되어 있는 접속자의 수나 네트워크의 응답시간 등은 고려하지 않고 차례대로 서버에게 작업을 할당하는 방식이다. 이 방식은 서버의 수행능력 상태나 네트워크의 상태를 고려하지 않기 때문에 실제 서버사이에 동적인 부하 불균형이 심각해질 수 있다.

2.3.2 최소 접속(Least-Connection)

최소 접속 방식은 서비스를 사용하는 접속자의 수가 가장 적은 서버로 요청을 직접 연결하는 방식으로 각 서버에서 동적으로 실제 접속한 사용자의 연결개수를 파악하여 스케줄링 하는 알고리즘이다. 이 방식은 HERMES 시스템의 경우 Broker서버에 접속해 있는 접속자의 수와 무관하게 Broker와 Provider서버간의 접속 상태가 좋지 않다면 부하를 분산하는데 불균형을 초래할 수 있다.

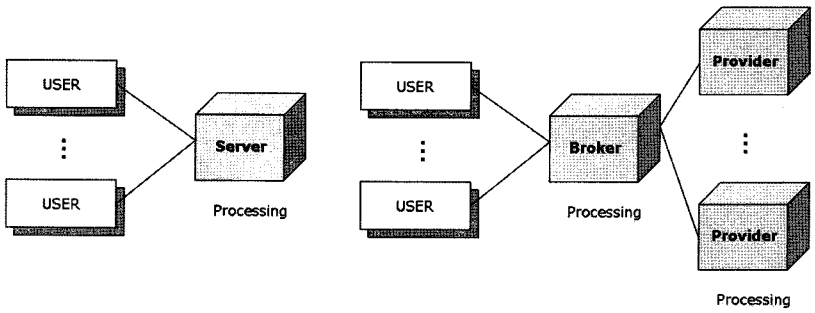
2.3.3 가중치 기반 라운드로빈(Weighted Round-Robin)

가중치 기반 라운드로빈 방식은 실제 서버에 서로 다른 성능 가중치를 부여하여 Load Balancing을 하는 방식이다. 각 서버에 가중치를 임의로 부여할 수 있으며, 여기서 지정한 가중치 값을 통해 처리 순서를 정한다. 이 방식은 요청에 대한 부하가 지속적으로 발생할 경우 불균형 상태를 초래할 수 있다.

2.3.4 가중치 기반 최소 접속(Weighted Least-Connection)

가중치 기반 최소 접속 방식은 가중치 기반 라운드로빈 방식과 마찬가지로 실제 서버에 서로 다른 성능 가중치를 부여하며, 접속자수를 고려하여 가중치를 부여한다.

각각의 방식들은 분산 환경에서의 Load Balancing을 위해서 일반적으로 사용되는 스케줄링 기법들이지만 HERMES 시스템의 경우 단순히 분산되어 있는 서버를 정해진 순서대로 스케줄링하거나 서버에 접속한 접속자의 수만을 가지고 스케줄링 하는 기법들로는 Broker서버를 균형 있게 Load Balancing할 수 없다. HERMES 시스템은 Query를 요청받는 Broker서버가 사용자의 Query를 직접 처리하지 않고 Query에 맞는 적당한 Provider서버들을 선택하기 위하여 최적화 작업을 마친 후 최적화된 Query에 적합한 Provider서버로의 Web Services요청만을 하기 때문이다. 따라서 Broker서버의 상태와 더불어 Broker와 Provider서버 간의 네트워크 트래픽 상태(Web Services 호출-응답 시간)를 고려하여 스케줄링 하는 방식이 필요하다. 또한 기존의 스케줄링 방식처럼 모든 서버들을 스케줄링에 참여 시키지 않고 부하 분산을 위한 Broker서버 목록 테이블을 만들어 Broker서버 전체의 평균적인 수행능력과 각각의 Broker의 수행능력을 비교하여 성능에 미치지 못하는 Broker서버를 스케줄링 작업에서 제외시킨다. 다음 스케줄링 작업이 시작될 때 제외되었던 Broker서버는 무조건 포함시킴으로써 수행능력을 비교할 때마다 Broker서버가 계속적으로 줄어드는 것을 방지한다. 이러한 과정을 통



(a) (b)

그림 4 일반 서버(a)와 HERMES 서버(b)

해서 선택된 Broker서버들의 성능에 따라서 가중치를 부여하고 라운드로빈(Round Robin) 방식을 사용하여 분산 처리함으로써 요청되는 Query들에 대해서 서버의 안정성과 응답시간에 대한 신뢰성을 제공한다. 분산 환경에서의 사용자 요청에 대한 처리 구조를 일반적인 서버와 HERMES 시스템을 구성하는 서버와 비교하여 보면 그림 4와 같다.

2.4 .Net Remoting

.Net Remoting이란 Microsoft의 서로 다른 호스트의 정보교환 기술로 클라이언트 프로그램이 원격 객체의 메서드를 지역 객체의 메서드처럼 호출 할 수 있게 해주는 기술이다.

2.4.1 원격 객체(Remote Object)

원격 객체는 서버에서 작동하는 객체를 말한다. 클라이언트는 이 객체의 메서드를 직접 호출하지 않고 대신 프록시를 사용한다. 원격 객체는 두 가지 형식으로부터 파생이 되는데MarshalByValueObject로부터 파생된 모든 클래스는 자신의 응용프로그램 도메인을 벗어나지 않기 때문에 지역 객체로 사용되고, MarshalByRefObject로부터 파생된 모든 클래스는 프록시를 사용하여 접근할 수 있는 원격 객체로 사용된다. 즉 클라이언트는 프록시를 생성하여 원격 객체를 자신의 객체처럼 호출한다.

2.4.2 채널(Channel)

채널이란 클라이언트와 서버 사이에 메시지가 전송되는 방법으로 클라이언트 부분과 서버부분으로 나누어져 있다. 대표적으로 HTTP채널과 TCP채널이 사용되고, 개발자가 다른 프로토콜을 이용하여 통신하는 사용자 지정 채널도 생성할 수 있다.

2.4.3 포매터(Formatter)

포매터란 메시지가 전송될 때 직렬화(Serialization) 시키는 방법으로 채널을 통하여 전송되는 형태를 말한다. 기본적으로 HTTP채널을 사용할 경우 XML 형식인 SOAPFormatter를 사용하고, TCP채널을 사용할 경우 Binary 스트림 형식인 BinaryFormatter를 사용한다. SOAPFormatter는 .Net 프레임워크에 기반을 두지 않는 Web Services와 통신하기 위해 사용되고, Binary-Formatter는 성능이 중요한 인트라넷 환경에서 사용된다.

2.5 Web Services 아키텍처

Web Services는 HTTP와 XML을 접목시킨 SOAP (Simple Object Access Protocol)을 이용하여 원격 메서드를 호출할 수 있는 방법이다. 주요 구성요소는 다음과 같다.

2.5.1 SOAP

원격 응용프로그램 간의 정보를 교환하는데 필요한 구조를 표준화한 것으로 XML문서 형태를 가지고 있고, 전송 프로토콜로서 운반되기 위한 방법에 대해서 언급한다.

2.5.2 WSDL

표준화된 방식으로 Web Services의 인터페이스를 기술하는 XML 기술이다. WSDL은 Web Services 시스템이 제공하는 원격 프로시저를 이용하기 위해 매개변수(인자), 프로시저(메서드)명 그리고 반환형을 기술하는 방법을 표준화한 것이다. Web Services 클라이언트는 WSDL 문서를 이용해서 Web Services 시스템에 바인딩 하는 방법을 알게 된다.

2.5.3 UDDI

Web Services의 공개(publish)와 검색(find)를 위한 XML 레지스트리(XML저장소)의 구현과 사용 방법을 표준화한 것이다. UDDI 표준은 Web Services 중개자가 운영해야 할 XML 레지스트리 기능과 Web Services 제공자가 어떻게 공개해야 하는지, 그리고 Web Services 소비자가 어떻게 검색할 수 있는지에 대해서 언급하고 있다[14].

3. 아키텍처

본 장에서는 기존에 연구된 HERMES 시스템의 확장

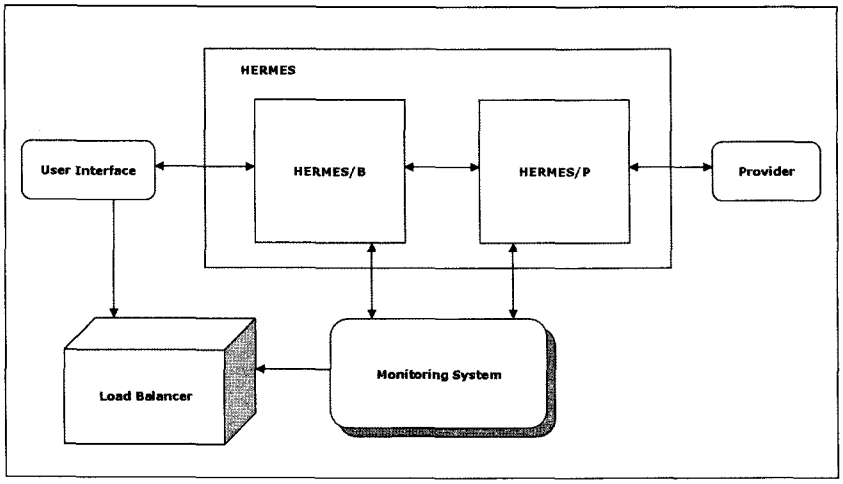


그림 5 Load Balancing 시스템 아키텍처

성과 다수 사용자의 요청에 대한 서버의 안정성을 보장하기 위하여 시스템의 특징에 맞는 부하 분산 스케줄링 기법을 적용한 아키텍처를 제안하고, Load Balancing 시스템을 이루고 있는 주요 컴포넌트들의 특징과 기능을 알아본다. 제안하는 Load Balancing 시스템 아키텍처는 그림 5와 같다.

3.1 Load Balancing 시스템의 구성요소

Load Balancing 시스템의 주요 컴포넌트 동작 과정을 보면 그림 6과 같다.

3.1.1 Collect_B_Info 컴포넌트

Monitoring System은 각 Broker서버의 컴포넌트 수행시간과 Broker와 Provider서버간의 Web Services의 호출-응답시간을 각각의 log파일로 저장한다. Load Balancer는 Collect_B_Info컴포넌트를 통하여 관리자가 정해놓은 주기마다 Monitoring System으로부터 저장된 파일을 전송받는다.

3.1.2 LOG 컴포넌트

Monitoring System으로부터 주기적으로 수집된 각각의 Broker서버 요구-응답시간을 저장한 파일

3.1.3 GetBrokerList 컴포넌트

GetBrokerList 컴포넌트는 우선순위를 결정하는 Broker서버의 목록 테이블을 만드는 컴포넌트로써

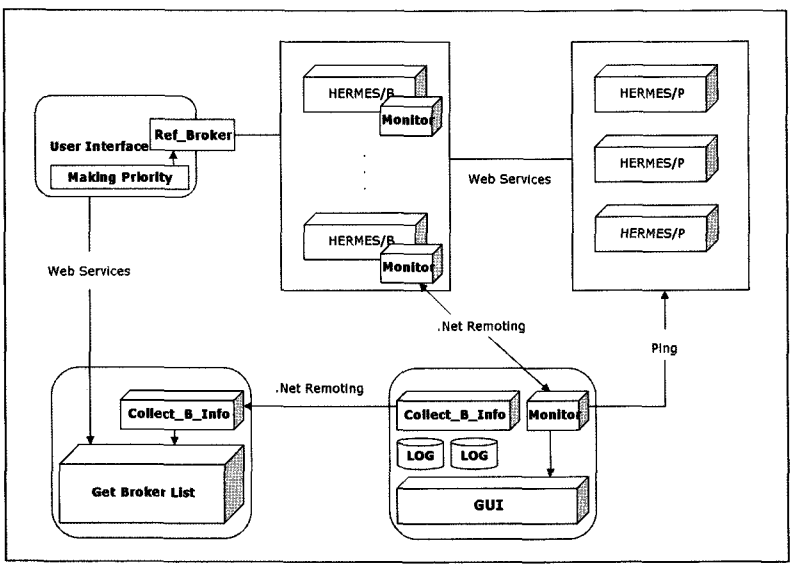


그림 6 Load Balancing 시스템의 주요 컴포넌트

Monitoring System으로부터 받은 각 Broker서버에 대한 요구-응답시간 파일을 열어서 전체 수행시간을 전체 Query의 개수로 나눈 평균 수행시간을 계산하고, 각 Broker서버의 평균 수행시간을 모두 더하여 Broker서버의 개수로 나눈 값을 구한다. 본 논문에서는 계산 된 값을 Broker서버의 부하 분산 스케줄링에 포함시키기 위한 Threshold 값으로 사용한다. 정리하면 다음과 같다.

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

Threshold 값과 Broker서버의 평균 수행시간을 비교하여 Broker서버의 값이 Threshold 값보다 작으면 스케줄링에 포함시키고 그렇지 않으면 제외시킨다. 다음 주기도 마찬가지로 스케줄링을 하기 위한 Broker서버를 선택할 때에는 동일한 방법으로 수행되며, 이전 주기에 제외되었던 Broker서버는 무조건 스케줄링에 참여시킨다.

3.1.4 Making Priority 컴포넌트

User Interface에서 접속자의 Query를 처리할 때 Broker서버를 정하기 위해서 Load Balancer의 GetBrokerList로부터 목록 테이블을 가져오는 컴포넌트이다. Broker서버의 목록 테이블에서 우선순위에 따라 원하는 Broker서버를 결정한 후 웹 참조 변수인 Ref_Broker의 URL을 갱신시킨다.

3.2 컴포넌트들 간의 협력관계

3.2.1 User Interface

접속자의 Query를 처리하기 전 Load Balancer의 GetBrokerList()로부터 가져온 Broker서버의 목록을 참조한다. Broker서버의 정보에 대한 업데이트 주기가 되면 Load Balancer의 Web Services를 사용하여 GetBrokerList()로부터 새로운 Broker서버의 목록을 가져온다.

3.2.2 Load Balancer

Monitoring System과 원격 객체인 Collect_B_Info객체를 사용하여 각 Broker서버의 저장된 데이타 파일들을 가져오기 위해 통신하고, 데이타 파일들을 열어서 Broker서버의 목록 테이블을 만들기 위하여 User Interface와 Web Services로 통신한다.

3.2.3 Monitoring System

Broker와 Provider서버의 상태를 감시하고 Load Balancer에서 Broker서버의 목록을 만들기 위해 필요한 정보를 Notify Manager객체를 통하여 수집한다.

Load Balancing 시스템의 전체적인 컴포넌트들과의 협력관계를 다이어그램으로 나타내면 그림 7과 같다.

3.3 Load Balancer 구현기술

3.3.1 Broker서버의 정보 수집을 위한 원격 객체

Collect_B_Info객체는 TCP Channel과 BinaryFormatter를 사용하여 서비스를 등록한다. 주요 코드를 보면 다음과 같다.

```
TcpServerChannel tcpCh = new TcpServerChannel(9000);
ChannelServices.RegisterChannel(tcpCh);
RemotingConfiguration.ApplicationName =
"BrokerInfoService";

RemotingConfiguration.RegisterWellKnownServiceType
(typeof(BrokerInfoManager.BrokerInfo),
"BrokerInfoService", WellKnownObjectMode.Singleton);
// Server측 코드
```

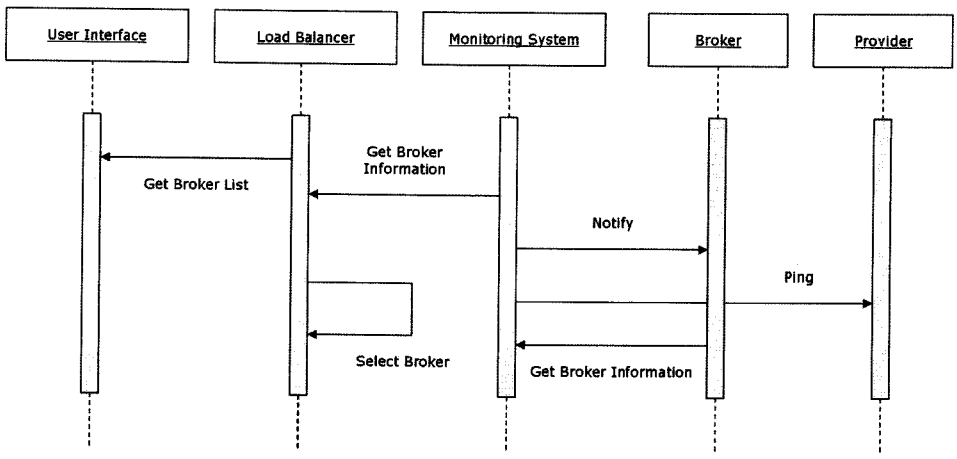


그림 7 Load Balancing 시스템의 시퀀스 다이어그램

```
ChannelServices.RegisterChannel(new
TcpClientChannel());
BrokerInfo obj = (BrokerInfo)Activator.GetObject
                 (typeof(BrokerInfo),
            "tcp://maple.hanyang.ac.kr:9000/BrokerInfo");
if(obj == null) {
   Console.WriteLine("could not locate server");
   return;
}
// Client측 코드
```

TcpServerChannel이나 TcpClientChannel클래스를 사용하기 위해서는 System.Runtime.Remoting.dll 어셈블리를 참조해야만 한다. Server측 코드에서 포트 번호 9000을 가진 System.Runtime.Remoting.Channels.Tcp.TcpServerChannel 클래스의 인스턴스를 생성한다. 이 채널을 원격 객체를 위한 채널로 사용할 수 있도록 System.Runtime.Remoting.Channels.ChannelService 클래스를 이용하여 등록한다. 이때 원격 객체 내의 클래스 형식, 클라이언트에 의해 사용되는 URL, 모드를 지정한다.

3.3.2 Broker서버 목록 테이블을 제공하는 Web Services

Collect_B_Info객체를 통하여 얻어온 데이타로 우선순위를 정한 Broker서버의 목록 테이블을 만든 코드를 보면 다음과 같다.

```
// 모든 Broker서버들의 요구-응답시간의 평균을 구한다.
int LValue = (danKookBorkerCValue + hy_110CValue
              + hy_91CValue) / brokerNum
// 평균값과 각 Broker서버들의 요구-응답시간을 비교한다.
int count = 0, j, temp;
for( int i=0 ; i < brokerNum ; i++){
   if(brokerInfo[i] > LValue) {
      count++;
   }
}
 int[] rankArray = new int[count];
  for(i=0 ; i < brokerNum ; i++) {
   if(brokerInfo[i] <= LValue) {
      rankArray[j] = brokerInfo[i];
      j++;
   }
}
// Broker서버를 수행능력이 좋은 순서대로 정렬
for(i=0 ; i < count-1 ; i++) {
   for(j=i+1 ; i<count ; j++) {
      if(rankArray[i] > rankArray[j] {
         rankArray[i] = temp;
         rankArray[i] = ranArray[j];
         rankArray[j] = temp;
      }
   }
}
```

Load Balancer는 웹 메서드인 GetBrokerList()를 통하여 Broker서버 목록 테이블과 관련된 처리 부분을 User Interface에게 노출시킴으로써 제공한다. 그림 8과 9에서 구현 기술들의 관계와 우선순위와 관련된 Web Services 부분을 확인할 수 있다.

4. 성능 분석

실험 환경은 분산되어 있는 Broker서버 3대와 Provider서버 10대를 이용하여 임의의 Query를 일정 시간 동안 수행시켜 다음 네 가지 경우로 비교했다. 각 서버는 최대 100Mbps의 WAN망으로 분산되어 있기 때문에 네트워크 트래픽에 의한 지연은 고려하지 않는다.

- 1대의 Broker서버를 사용한 경우
- 3대의 Broker서버를 라운드로빈(Round Robin) 스케줄링 방법을 적용한 경우
- 3대의 Broker서버를 최소접속(Least Connection) 스케줄링 방법을 적용한 경우
- 3대의 Broker서버를 Broker서버 목록을 이용한 수정된 라운드로빈(Round Robin) 스케줄링 방법을 적용한 경우

Query의 요구-응답시간을 비교한 그래프와 평균 요구-응답시간을 비교한 그래프를 살펴보면 Load Balancing을 적용한 Broker서버가 전체적으로 요구-응답시간 면에서 좋은 성능을 보였고, Broker서버 목록을 이용한 수정된 라운드로빈(Round Robin) 스케줄링 방법이 라운드로빈(Round Robin) 스케줄링이나 최소접속(Least Connection) 방법보다 더 좋은 성능을 보이는 것을 그림 10과 그림 11에서 확인할 수 있다.

그림 12부터 15는 Monitoring System을 통하여 Broker서버의 어떤 컴포넌트가 수행되는지를 확인하고 그 컴포넌트의 수행시간을 측정하여 보여준다. 정해진 주기마다 측정된 값을 가지고 Load Balancer로부터 Broker서버의 스케줄링이 일어나고 Query를 처리하는 Broker 서버가 바뀐 것을 확인할 수 있다.

5. 결론 및 향후 과제

본 논문은 기존의 HERMES 시스템의 Broker서버에 다수의 동시 접속자들이 Query를 수행할 경우 발생할 수 있는 Overhead를 줄이기 위해서 Broker서버를 증설하고 Load Balancing 하기 위한 시스템을 구축하였다. 또한 Broker서버의 내부 컴포넌트 수행시간과 Broker와 Provider서버간의 Web Services 호출에 의한 네트워크 트래픽 시간을 함께 측정하여 Broker서버의 우선순위를 결정할 수 있는 스케줄링 방법을 연구하였다.

실험을 통하여 분산 환경에서의 대표적인 부하 분산 스케줄링 방법인 라운드로빈(Round Robin)과 최소 접

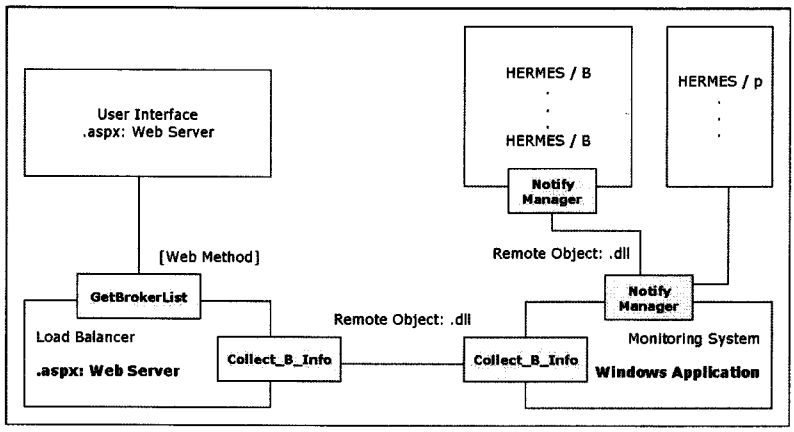


그림 8 원격 객체와 Web Services의 호출관계

그림 9 Load Balancer의 UI와 협력하는 Web 메서드

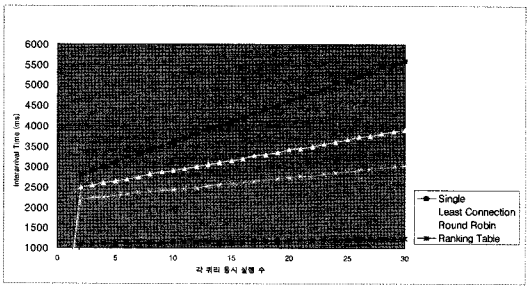


그림 10 각 Query 동시 실행 시 평균 Inter arrival Time

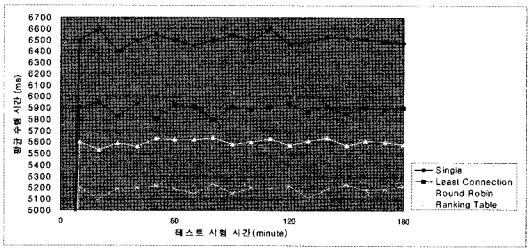


그림 11 해당시간동안 발생한 Query들의 평균 수행 시간

속(Least Connection) 방법을 본 논문에서 제안한 방법과 비교한 결과 제안한 방법이 평균적으로 더 좋은 성능을 보이는 것을 실험 결과 그래프를 통하여 확인하였다. 또한 일반적인 클라이언트-서버 분산 시스템에서는

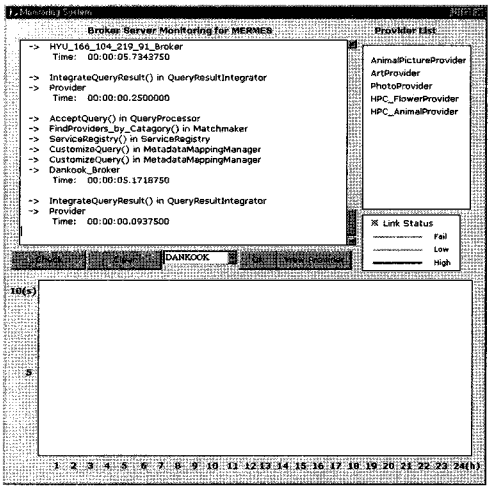


그림 12 Broker서버의 스케줄링 화면(a)

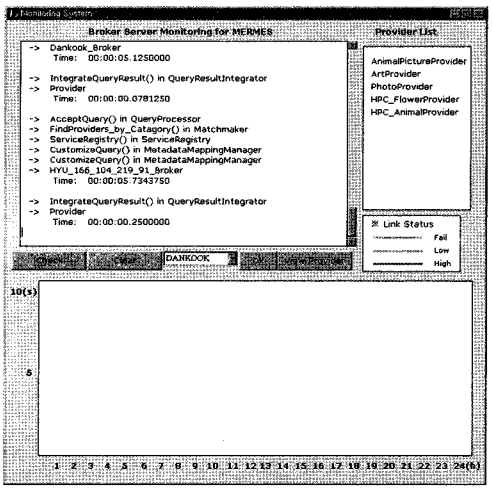


그림 13 Broker서버의 스케줄링 화면(b)

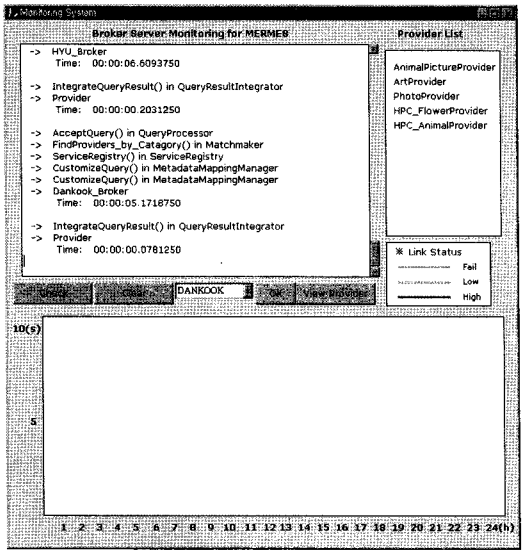


그림 14 Broker서버의 스케줄링 화면(c)

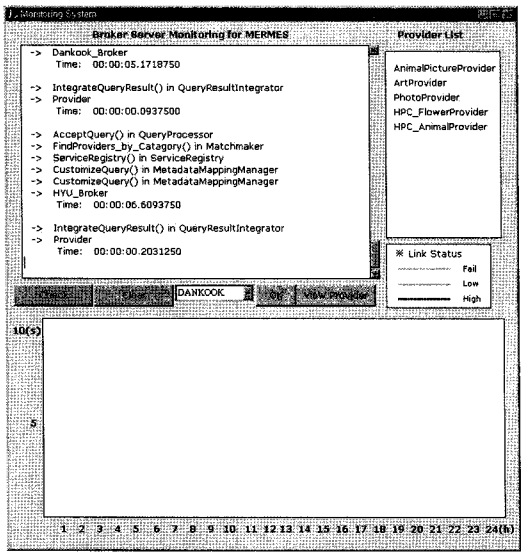


그림 15 Broker서버의 스케줄링 화면(d)

서비스를 직접 처리하는 서버의 상태가 Load Balancing을 위한 중요한 Metric이 될 수 있었지만 HERMES 시스템의 Broker서버의 경우 서비스를 처리하는 Provider서버와의 네트워크 상태 또한 Broker서버의 상태만큼 중요하다는 것을 확인하였다. 본 논문의 Load Balancing 시스템의 주요 컴포넌트들은 Web Services 기술로 구현되었기 때문에 이와 같은 구조를 갖는 다른 플랫폼의 시스템에서도 컴포넌트들의 정보만 URL로 가져온다면 언제든지 사용가능하다.

하지만 현재 HERMES 시스템의 Broker서버는 내부 컴포넌트에 대한 멀티 쓰레딩이 지원되지 않고 있기 때문에 다수의 Query가 Provider서버로의 Web Services 호출이 발생하기 전까지는 다음 Query들이 작업 큐에서 대기해야된다. 물론 Broker서버는 사용자가 원하는 서비스를 직접 처리 하지 않기 때문에 일정양의 Query에 대해선 문제가 없겠지만 다수의 접속자와 시스템이 확장될 수 있다는 것을 고려한 만큼 앞으로 연구되어야 할 과제가 된다. 또한 Provider서버에 대한 Web Services호출이 빈번하므로 Provider서버가 등록되어 있는 UDDI서버의 스키마를 사용한 Web Services의 QoS 평가가 연구된다면 성능과 신뢰성이 더욱 향상될 것이다.

참 고 문 헌

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

[2] 나연묵, 이복주, 김정선, "Visual Media Retrieval Framework Using Web Service," Lecture Notes in Computer Science 3597권, 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://linuxvirtual-server.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.

심 준 용

2005년 2월 우석대학교 컴퓨터공학과 졸업(학사). 2007년 2월 한양대학교 컴퓨터공학과 졸업(석사). 2007년 1월~현재 LIG넥스원(주) 시스템연구소 근무(연구원). 관심분야는 미들웨어, 상호운용 아키텍처, 분산객체 기술

원 재 훈

2006년 2월 한양대학교 화학공학과 졸업(학사). 2008년 2월 한양대학교 컴퓨터공학과 졸업(석사). 관심분야는 객체지향, 분산 컴퓨팅

김 세 창

2006년 2월 한양대학교 컴퓨터공학과 졸업(학사). 2008년 2월 한양대학교 컴퓨터공학과 졸업(석사). 관심분야는 객체지향, MHP 미들웨어

김 정 선

1986년 서울대학교 컴퓨터공학과 졸업(학사). 1988년 Iowa State University 전기및컴퓨터공학과 졸업(석사). 1994년 Iowa State University 전기및컴퓨터공학과 졸업(박사). 1994~1996년 한국전자통신연구원(ETRI) 선임연구원. 1996년~현재 한양대학교 전자컴퓨터공학부 부교수. 관심분야는 Parallel/Distributed Processing, Distributed Object Computing, Component Based Development