論文2002-39CI-6-1

이동 에이전트 기반의 웹 서버 부하 모니터링 시스템 구현

(An Implementation of Web Server Load Monitoring System Based on the Mobile Agent)

朴 洪 珍 *

(Hong-Jin Park)

요 약

인테넷이 급격히 발전함에 따라 인터넷 사용자가 크게 증가하고 있으며, 이에 따른 웹 서버의 중요성도 증가되고 있다. 인기있는 웹 사이트는 사용자의 집중현상으로 웹서버의 부하가 크게 증가하여 오류가 발생하거나 웹서비스가 중단되는 현상이 발생하기도 한다. 따라서 올바른 웹서비스를 제공하며 웹서버의 성능 관리를 위해서 웹서버의 부하를 효율적으로 모니터링하는 기술이 필수적이다. 기존 클라이언트-서버 방식을 이용한 웹 부하모니터링 기법은 서버 프로그램을 모든 웹 서버에 미리 설치해야하는 문제점이 있다. 본 논문은 클라이언트 서버 방식의 문제점을 해결하기 위해 이동 에이전트를 이용한 웹 서버 부하 모니터링 시스템을 구현한다. 이동 에이전트를 기반하여 구현된 시스템은 미리 프로그램을 설치할 필요가 없으며 네트워크 트래픽을 줄이고 웹서버의 추가적인 부하를 줄일수 있다.

Abstract

According to the rapid evolution of internet, the number of internet users has greatly increased, so that the web servers are getting more important. If there are a great number of users trying to connect to a popular web server, the load of the web server will increase rapidly, so that failures may occur or web services may stop. An efficient load monitoring technique is required for the available web services and the performance management. The traditional techniques for load monitoring of web server using the client-server method have the problem that the server programs in client-server method must be pre-installed to all web servers. This paper implements web server load monitoring system using the mobile agent in order to overcome the problem of client-server computing. The system based on the mobile agent has no need to install the program in advance and it reduces the network traffic and the additional overheads in web servers.

Keywords : Mobile Agent, RMI, Web Server, Load Monitoring

1. 서 론

인터넷 사용자의 급속한 증가에 따라 신뢰성 있는 웹서버의 중요성이 크게 부각되고 있다. 특히, 인기 사이트는 사용자의 집중현상으로 웹서버의 부하가 짧은 시간안에 크게 증가하여 오류가 발생하거나 웹서비스가 중단되는 현상이 발생하기도 한다. 과중한 부하를 분산시키기 위해 미러링 웹 서버를 이용하나, 네트워크

* 正會員, 尙志大學校 컴퓨터情報工學部
(School of Computer, Information and Communication Engineering, Sangji University)

※ 이 논문은 2001년도 상지대학교 연구비 지원에 의한 것임.

接受日字:2002年6月20日, 수정완료일:2002年10月14日

상태, CPU 사용률, 메모리, 디스크의 부하 정도를 고려하지 않고 웹 서버를 선택하기 때문에 부하가 집중되는 현상을 발생시킬 수 있다. 따라서 올바른 웹서비스를 제공하고 웹서버의 성능 관리를 위해서 웹서버의 부하를 효율적으로 모니터링하는 기술이 필수적이다[1-3].

기존의 부하 모니터링에 사용한 분산 컴퓨팅 방법은 RPC(Remote Procedure Call), CORBA(Common Object Request Broker Architecture), Java RMI (Remote Method Invocation)와 같은 클라이언트-서버 파라다임이다. 클라이언트-서버 파라다임에서 서버는 시스템의 자원에 대한 사용을 허가함으로써 서비스를 제공하며, 웹 서버 마다 부하를 측정하는 서버 프로그램을 미리 설치해야한다. 또한, 클라이언트와 서버간 부하 정보 전송하기 위해 각 요청 메시지마다 응답 메시지를 전송해야하며 이는 네트워크 트래픽을 증가시키는 요인이 된다.

본 논문은 이를 해결하기 위해 이동 에이전트 기반의 웹서버 부하 모니터링 시스템을 설계 및 구현한다. 이동 에이전트는 처음 생성된 시스템에 제한되지 않고, 자유롭게 네트워크상의 다른 호스트로 이동할 수 있는 프로그램이며, 클라이언트-서버 파라다임보다 네트워크 부하를 감소시키며, 자율적으로 실행할 수 있는 장점이 있다[4]. 이동 에이전트의 순회(itinerary) 이동 패턴과 주-종속(master-slave) 작업 패턴을 결합하여 설계한 웹 서버 부하 모니터링은 웹서버의 부하 증가를 판단할 수 있는 중요한 부하 측정 요소를 모니터링 작업을 이동 에이전트에 추가하였다. 기존의 클라이언트-서버 방식의 부하 모니터링 기법보다 제안된 이동 에이전트 방식은 스스로 시스템을 순차적으로 이동하면서 부하 모니터링 작업을 수행하기 때문에 웹서버마다 미리 프로그램을 설치할 필요가 없으며 네트워크 사용량을 감소시키며 웹서버에 추가적인 부하 부담을 감소시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 웹 서버 부하 모니터링의 필요성을 설명한후 3장에서 기존 부하 모니터링 방식인 클라이언트-서버를 이용한 부하 모니터링의 문제점 기술한다. 4장과 5장에서는 본 논문에서 제시하고 있는 이동 에이전트 기반의 부하 모니터링 시스템의 설계와 구현에 대해 설명한다. 6장에서는 RMI 방법을 이용한 부하 모니터링 시스템과 이동 에이전트를 사용한 부하 모니터링 시스템의 성능 분석 결과를 설명하고, 마지막으로 7장에서 결론을 맺는다.

Ⅱ. 웹서버 부하 모니터링의 필요성

웹서버가 사용자수, 전송량, 프로세스 수, 메모리 사용량의 급증으로 인해 그 부하가 증가하거나, 시스템 자체에 문제가 발생하게 되면 웹서비스를 올바로 제공할 수 없으며 이는 많은 사용자들에게 큰 오류로 인식되어 웹서버 개발비용을 낭비하는 결과를 초래한다[3, 13]. 또한 웹서버 관리자가 여러 서버에 발생하는 오류를 지속적으로 검사하고, 통합적으로 관리하기가 어려우므로 자동으로 웹서버의 부하를 측정해서 문제 발생을 알려주는 기술이 필요하게 된다[5].

따라서, 웹서버를 운영하는데 필요한 모니터링 기능은 먼저 웹 사이트의 모든 컴포넌트가 올바로 작동하고 있는지를 지속적으로 검사해야하며, 오류가 발생했을 때 이것을 짧은 시간내에 검출할 수 있어야 한다[3]. 그리고 웹서버의 사용자가 급증할 때를 대비해서 웹서버에 접속하는 사용자 수와 자료의 전송량이 급증하는 것을 감지해서 시스템이 효율적으로 운영될 수 있도록 최적화 작업을 수행함으로서 자원의 과다한 사용에 따른 오류 발생을 예방할 수 있다. 웹서버의 부하를 주기적으로 측정한 후 DB에 저장해 놓은 로그 기록(historical data)을 사용해서 시스템 성능을 분석하고 추후 시스템 네트워크 사용률, 최고 트래픽 시간을 예측할 수 있으며 시스템 확장의 근거 자료로 활용할 수 있다[6].

웹서버의 부하를 나타내는 중요한 요소는 네트워크

표 1. 웹서버의 부하 요소
Table 1. Load factors of the web server.

부하 요소	중요성
네트워크 연결	서버는 사용자들이 접속할 수 있도록 항상 네트워크에 연결되어 있어야 한다.
자료 전송량	각 사용자들에게 전송하는 자료의 양이 많아질수록 네트워크 사용량이 증가해 전송 속도가 느려진다.
메모리 사용률	메모리 사용량에 따라 서버의 처리 속도가 영향을 받게된다.
가용 디스크 량	가용 디스크의 양이 어느 수준 이하로 떨어지게 되면 디스크 풀 오류의 위험이 예상된다.
디스크 사용률	디스크의 사용률이 높을수록 디스크 억세스 속도가 느려진다.

연결, 자료 전송량, CPU 사용률, 메모리 사용률, 디스크 사용률이며 이러한 부하 요소의 중요성은 <표 1>과 같으며, 이러한 요소의 값을 정확하게 측정해야 한다.

Ⅲ. 기존 웹 서버 부하 모니터링 방식

기존의 웹서버의 부하를 모니터링 하기 위해서는 주로 RPC, CORBA, Java RMI와 같은 클라이언트-서버 방식을 사용하였다[7-8]. 클라이언트-서버 패러다임은 모든 웹서버마다 부하를 측정하는 서버 프로그램을 설치하고, 부하 정보를 수집하고자 하는 시스템에는 웹서버에게 부하 정보를 요청하는 클라이언트 프로그램은 설치해야만 한다. 각 웹서버에 설치한 서버 프로그램은 주기적으로 클라이언트의 요청에 대한 응답으로 부하 정보를 전송해 주어야 한다(<그림 1> 참조)[9-10].

클라이언트-서버 프로그램은 부하 정보를 전송하기 위해 <그림 2> 처럼 풀(pull) 프로토콜이나 푸쉬(push) 프로토콜을 사용한다[12]. 풀 프로토콜은 클라이언트가 정보가 필요할 때마다 요청 패킷을 보내면 서버가 원하는 정보를 전송해 주는 방법이다. 풀 프로토콜에서는 매번 요청 패킷을 전송해야하기 때문에 네트워크 대역폭을 낭비하게 되며 네트워크 지연 시간도 커진다. 클라이언트는 서버의 상태를 고려하지 않고 정보를 요청하게 되는 오류를 범할 수 있다.

푸쉬 프로토콜은 서버가 일방적으로 클라이언트에게 일정한 시간 간격으로 정보를 전송해 주는 방법이다. 매번 요청 패킷을 낭비하지 않으므로 전송량과 지연

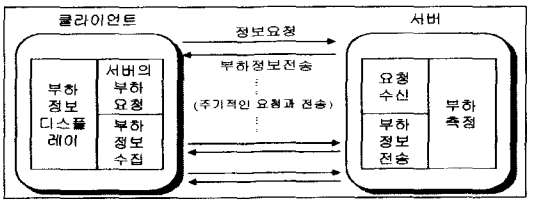


그림 1. 클라이언트 서버 방식의 부하 모니터링
Fig. 1. Load monitoring in client-server.

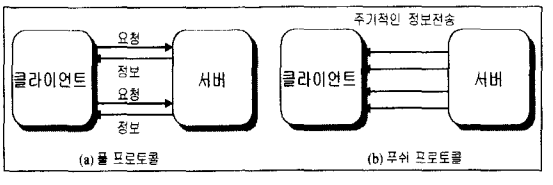


그림 2. 풀 프로토콜과 푸쉬 프로토콜
Fig. 2. Pull protocol and push protocol.

시간은 줄어들지만, 서버는 클라이언트가 정보를 받기 원하는 시간을 맞추기 어려우며 클라이언트가 정보를 받을 수 있는 상태가 아닐 수도 있다. 클라이언트가 원하지 않을 때는 받은 정보를 버리게 되며 이는 그만큼의 정보 전송량 낭비의 결과를 초래한다. 그 후에 클라이언트가 원하는 정보를 받으려면 서버가 재전송 할 때까지 기다려야 하므로 지연시간이 발생한다.

기존의 웹 부하 정보를 모니터링 하기 위해 사용된 클라이언트 서버 방식은 다음과 같은 문제점이 있다. 첫째, 클라이언트와 서버간의 요청, 응답 메시지 전송으로 인해 네트워크 트래픽을 증가시키며, 네트워크 연결이 항상 유지되어야 하며 여러 가지 환경 변화에 자동적으로 대처하기 어렵다[11]. 둘째, 웹서버에서 서버 프로그램은 계속적으로 부하를 측정하면서 클라이언트의 요청을 기다려야 하기 때문에 일정한 메모리와 CPU를 점유하면서 웹서비스에 치중하는 웹서버에 추가적인 부하 부담을 주게된다. 셋째, 부하를 모니터링 하려는 사용자가 한 노드에서 모니터링 방법에 변경을 가할 수 없으며, 지리적으로 분산된 웹서버의 성능 관리가 매우 어렵다. 이러한 문제점을 해결하기 위해 본 논문에서 이동 에이전트를 이용하여 웹 서버 부하를 모니터링 하고져 한다.

Ⅳ. 부하 모니터링 시스템의 설계

1. 이동 에이전트 기반의 부하 모니터링

웹서버의 부하를 모니터링 하기 위해 상태 정보와 클래스 코드 직접 목적지 노드에 이동되는 이동 에이전트은 기존 클라이언트-서버 보다 네트워크 부하를 감소시킬 수 있다. 또한, 이동 에이전트를 이용해서 모니터링의 대상이 되는 모든 웹서버에 별도의 프로그램을 설치하지 않고도 모든 웹서버의 부하 정보를 한 시스템에 수집할 수 있다. 이동 에이전트의 실행 코드가 네트워크 상에서 웹서버로 이동해서 시스템 내부적으로 부하를 측정하고 그 결과값을 모니터링 서버에 전송한 후에 다음 웹서버로 이동한다. 별도의 서버 프로그램을 설치해서 실행하지 않아도 이동 에이전트가 자동으로 이동하며 원하는 작업을 수행하기 때문에 각 웹서버에 계정이 없거나 관리자 권한이 없는 일반 사용자들도 자신이 원하는 웹서버의 부하 정보를 수집할 수 있다. 이동 에이전트는 부하 측정에 필요한 핵심적인 코드만 소유하고 있기 때문에 그 실행 코드의 크기

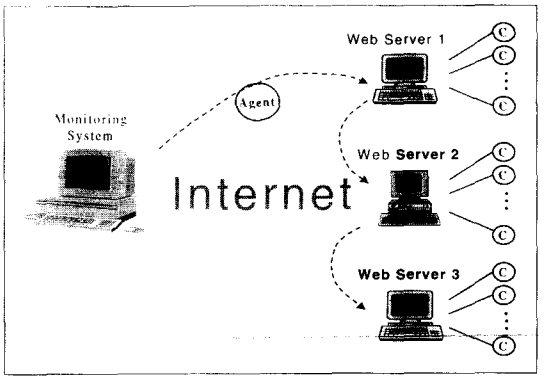


그림 3. 부하 모니터링 시스템의 구성
Fig. 3. Configuration of the load monitoring system.

가 아주 작은 크기로 구현될 수 있으며, 에이전트의 이동시에 필요한 전체 네트워크 사용량을 감소시킬 수 있다.

또한, 이동 에이전트는 웹서비스에 치중하는 웹서버에서 계속 실행되지 않고 에이전트가 이동했을 때에만 짧은 시간동안 실행되고 종료되기 때문에 웹서버에 추가적인 부하 부담을 주지 않는다. 웹서버의 부하를 측정하기 위해서는 큰 용량의 로그파일을 필요로 하지만 로그 파일을 네트워크를 통해 전송하지 않고 에이전트가 직접 이동해서 시스템 내부적으로 로그 파일을 빠른 시간 안에 분석해서 사용자수와 전송량을 계산해 낼 수 있다. 웹서버의 부하 모니터링 시스템에 이동 에이전트 기술을 적용한 전체적인 구성은 <그림 3>과 같다.

2. 이동 에이전트의 패턴 설계

부하 모니터링 시스템을 설계하기 위해서는 먼저 적합한 기능과 이동성을 가진 이동 에이전트를 설계해야 한다. 적합한 이동 에이전트는 순회 이동 패턴과 주-종속 작업 패턴을 결합하여 설계하였다.

2.1. 에이전트의 순회(Itinerary) 패턴 설계

여러 웹서버들의 부하 정보를 측정하는 이동 에이전트를 구현하기 위해서는 먼저 에이전트의 이동 패턴을 설계해야 한다. 부하 모니터링에 적합한 이동 패턴을 설계함으로서 에이전트가 여러 시스템을 이동하는 절차와 방법을 효율적으로 관리할 수 있다. 이동 에이전트가 여러 웹서버를 이동하면서 각 웹서버의 부하를 모니터링하기 위해서는 순회 패턴을 적용해야 한다. 순회 패턴은 여러 목적 호스트를 원하는 순서대로 이동하는 이동 패턴이다. 이것은 목적지의 주소 목록을 소유하고 있으며 이동 절차를 정의하고, 목적지가 존재하

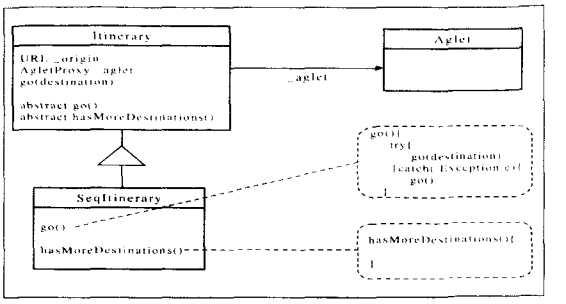


그림 4. 순회 패턴의 구조적 관계
Fig. 4. The structural relationship of itinerary pattern.

지 않는 예외의 경우에 대처하며 항상 다음에 어디로 이동 할 것인지를 알고 있다.

순회 패턴은 에이전트의 이동과 여정을 객체화한다. 에이전트는 여러 호스트를 스스로 이동하며 새로운 호스트로 이동할 때 발생할 수 있는 오류를 처리함으로서 자신의 여정을 동적으로 변경한다. 결국 에이전트의 행위와 메시지 처리에서 에이전트의 이동 처리를 분리함으로서 모든 부분을 모듈화 하였다. 순회 패턴의 가장 핵심적인 부분은 이동 처리를 에이전트 객체에서 Itinerary 객체로 옮기는 것이다. Itinerary 클래스는 에이전트의 여정을 변경하는 인터페이스와 새로운 호스트로 이동시키는 인터페이스를 제공한다.

에이전트 객체와 Itinerary 객체는 다음과 같은 방법으로 연결된다. 에이전트는 Itinerary 객체를 생성해서 이동할 목적지의 목록과 에이전트의 참조자를 가지고 초기화한다. 그 후에 에이전트는 go() 메소드를 사용해서 자신을 다음 목적지로 이동시킨다. 그러기 위해서는 Itinerary 객체는 에이전트와 에이전트의 참조자와 함께 전송되어야 한다.

순회 패턴을 사용함으로서 에이전트의 이동 절차를 에이전트의 행위와 분리시키며 각 부분을 모듈화 하였다. 또한 에이전트가 여러 호스트를 순서대로 이동하는 일정한 인터페이스를 제공한다.

<그림 4>는 순회 패턴을 이루는 클래스들의 구조적인 관계를 나타낸다. Itinerary 객체는 2가지의 추상 메소드(go(), hasMoreDestinations())를 사용해서 에이전트가 여러 호스트를 순차적으로 순회하는 방법을 제공한다. SeqItinerary 객체는 Itinerary 객체의 추상 메소드를 구현하며 에이전트의 현재 목적지 URL을 가진다.

<그림 5>는 Itinerary 객체와 에이전트간의 상호작용을 설명한다. 첫 번째 호스트에 존재하는 에이전트는

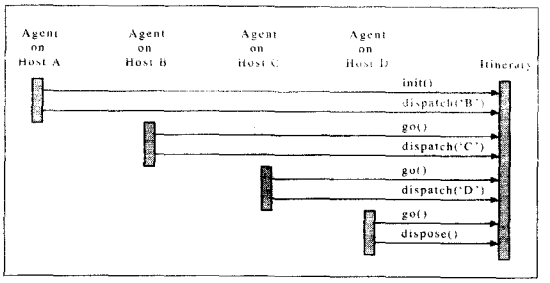


그림 5. Itinerary 객체와 에이전트의 상호작용
Fig. 5. Interaction between itinerary object and agent.

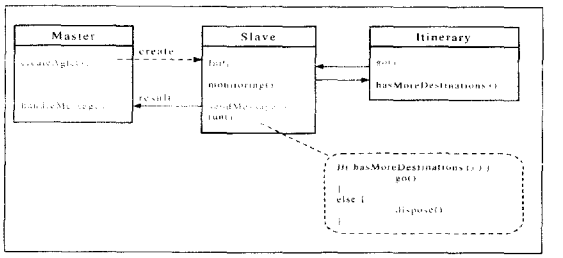


그림 6. 순회 패턴 기반의 주-종속 에이전트의 구조적 관계
Fig. 6. The structural relationship of master-slave agent based on the itinerary pattern.

Itinerary 객체의 함수를 다음과 같은 순서로 호출함으로서 여러 호스트를 순서대로 이동하게 된다.

SeqItinerary 객체는 에이전트에 의해 초기화된다. SeqItinerary 객체는 에이전트를 첫 번째 호스트로 이동시킨다. 에이전트가 Itinerary 객체의 go() 메소드를 실행시킬 때, 에이전트는 다음 호스트로 이동된다. SeqItinerary의 go() 메소드에서는 에이전트가 이동할 때 오류가 발생하게 되면 다음 호스트로 이동하게 하는 이동 패턴을 구현하였다.

순회 패턴을 사용함으로서 웹서버의 부하를 모니터링 하기 위한 에이전트의 이동 패턴을 쉽게 구현할 수 있으며, 목적지가 되는 여러 웹서버를 순차적으로 방문할 수 있다. 이러한 이동 기능을 구현하는데 있어서 에이전트 클래스를 수정할 필요가 없다.

2.2 에이전트의 주-종속(Master-Slave) 패턴 설계

주-종속 패턴은 주 에이전트가 수행하려는 작업을 종속 에이전트를 생성시켜 위임하는 방법을 사용한다. 주-종속 패턴을 사용하는 이유는 성능적인 면에서 효율적이기 때문이다. 주 에이전트는 종속 에이전트에 관계없이 계속 병렬식으로 다른 작업을 수행할 수 있다. 주 에이전트는 데이터를 입력하고 계속적으로 결과값을 출력하기 위한 GUI 기능을 제공할 수 있다. GUI와 모니터링 작업을 단일 에이전트가 제공한다면, 에이전트가 원격 호스트로 이동한 이후에 계속적으로 GUI 기능을 유지할 수 없다. 결국, 종속 에이전트가 이동한 이후에 하나의 주 에이전트가 GUI 기능을 제공해야 하며, 종속 에이전트는 원격 호스트에서 모니터링 작업을 수행하고 결과값을 전송해야 한다.

주-종속 패턴의 기본적인 개념은 Master, Slave 클래스를 이용하며, Master 클래스는 부하 모니터링 작업의 결과값을 수집하여 처리한다. Slave 클래스는 다른 호스트로 이동해서 부하 모니터링 작업을 수행하기 전에 초기화 작업을 수행한다. 에이전트가 수행하는 작업과 병렬식으로 다른 작업을 수행해야 할 필요가 있을 때와 고정 에이전트가 원격 호스트에서 작업을 수행할 필요가 있을 때 주-종속 패턴을 적용해야한다.

에이전트의 이동 방법을 정의하는 순회 패턴과 에이전트의 작업 수행 방법을 정의하는 주-종속 패턴을 결합함으로서 웹서버의 부하 모니터링에 적합한 에이전트를 설계하였다. 종속 에이전트는 순회 패턴을 사용해 여러 호스트를 이동하면서, 각 호스트에서 부하 모니터링 작업을 수행한다. 종속 에이전트는 주 에이전트가 생성한 Itinerary 객체를 매개 변수로 전달받아서 이동 방법을 제어한다. 부하 모니터링 작업을 모두 수행한 이후에는 다음 호스트로 이동하며 더 이상 이동할 호스트가 없을 때에는 스스로 소멸된다.

<그림 6>은 순회 패턴 기반의 주-종속 에이전트의 구조적 관계를 나타낸다. 주 에이전트는 createAglet() 메소드를 사용해 종속 에이전트를 생성시켜서 Itinerary 객체를 매개변수로 전달한다. 종속 에이전트가 보내온 결과값을 처리하는 handleMessage() 메소드를 구현하였다. 종속 에이전트는 Itinerary 객체를 사용해 원격 호스트를 순서대로 이동하면서 부하 모니터링 작업을 수행한다. 더 이상 이동할 호스트의 주소가 없으면 스스로 소멸된다.

2.3 부하 모니터링 시스템 설계

순회 패턴과 주-종속 패턴을 결합하여 설계한 부하 모니터링 시스템의 전체 구성은 <그림 7>과 같다.

<그림 7>에서 각각의 시스템에는 MAS(Mobile Agent Server)가 있다. MAS는 각각의 호스트에서 이동 에이전트가 실행될 수 있는 환경을 제공하며 이동 에이전트의 생성, 이동, 소멸과 같은 기능이 수행될 수 있다.

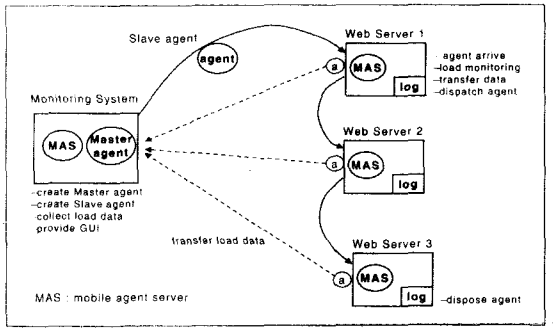


그림 7. 부하 모니터링 시스템의 설계
Fig. 7. Design of the load monitoring system.

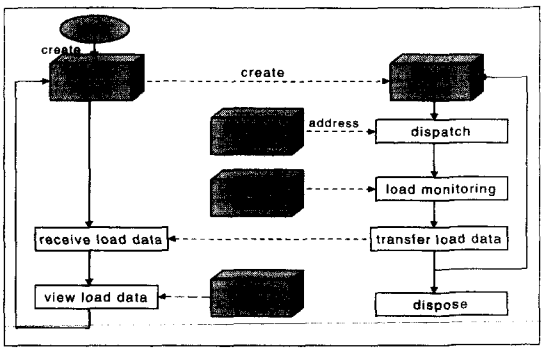


그림 8. 부하 모니터링 시스템의 클래스 구성
Fig. 8. Classes of the load monitoring system.

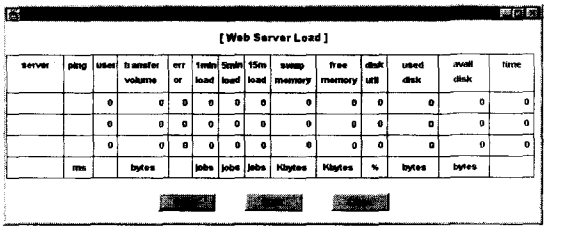


그림 9. Viewer 대화 상자
Fig. 9. Viewer dialog box.

에이전트는 이동 방법에 관계된 순회 패턴과 작업 수행과 관계된 주-종속 패턴을 결합하여 설계하였다. 모니터링 시스템의 MAS에서 Master 에이전트가 실행되며, Master 에이전트는 실제 웹서버로 이동할 Slave 에이전트를 생성한다. Slave 에이전트는 스스로 자신을 첫 번째 웹서버로 이동시켜서 웹서버의 로그 파일을 분석하고, 시스템 자체의 부하를 측정한 후에, 부하 정보를 Master 에이전트에게 전송한다. 부하 정보를 전송한 후에 다음 순서의 웹서버로 이동해서 같은 작업을 계속 수행한다. Master 에이전트는 모니터링 시스템에서 계속 실행하면서 GUI 기능을 제공하며 Slave 에이전트로부터 부하 정보를 받아서 사용자에게 전체 웹서버들의 부하 정보를 보여주고, 주기적으로 새로운 Slave 에이전트를 생성시킨다.

V. 부하 모니터링 시스템의 구현

이동 에이전트를 이용한 웹서버의 부하 모니터링 시스템의 전체 클래스 구성은 <그림 8>과 같다.

부하 모니터링 시스템은 Master 에이전트와 Slave 에이전트로 구성된다. Master 에이전트는 Slave 에이전트로부터 웹서버의 부하 정보를 받아서 전체 웹서버의 부하 정보를 Viewer 클래스를 사용해서 화면상에 보여주는 GUI 기능을 수행하며, 일정한 시간간격으로 Slave 에이전트를 새로 생성시켜서 모니터링 작업을 계속 수행한다.

Master 에이전트에 의해 생성된 Slave 에이전트는 스스로 그 자신을 웹서버로 이동시킨다. Slave 에이전트가 이동해야할 웹서버의 주소를 Itinerary 클래스에서 제공받는다. Itinerary 클래스는 모니터링의 대상이 되는 웹서버들의 주소를 가지고 있으며, Master 에이전트가 Slave 에이전트를 생성시킬 때 매개변수로 전달한다. 또한 Itinerary 클래스는 에이전트를 다음 주소로 이동시키고, 이동할 웹서버에 문제가 있을 때 그 다음 웹서버로 이동시키는 기능을 제공한다.

웹서버로 이동한 Slave 에이전트는 Monitor 클래스를 사용해서 웹서버의 로그를 분석하고, 시스템 자체의 부하를 측정한다. 모든 부하 정보를 Master 에이전트로 전송한 후에 다음 웹서버로 이동하며, 더 이상 이동할 웹서버가 없으면 모든 작업을 수행한 후에 스스로 소멸된다.

처음에 MAS상에서 Master 에이전트를 생성하면 <그림 9>와 같은 Viewer 대화상자가 나타난다.

웹서버의 부하를 모니터링하기 위해서는 웹서버의 주소와 에이전트의 시간간격을 입력해야 한다. Viewer 대화상자에 있는 Setup 버튼을 선택하면 <그림 10>과 같은 설정 대화상자가 나타난다.

설정 대화상자에서 Add 버튼을 사용하여 웹서버의 주소를 등록하거나, Delete 버튼은 사용하여 등록된 주소를 삭제한다. 모든 웹서버의 주소를 등록한 후에 최근의 사용량 검색 시간, 최근의 에러 검색 시간, 이동 에이전트의 시간간격을 입력한다. 모든 입력이 끝난 후 Save All 버튼을 선택하여 입력한 사항을 저장한다.

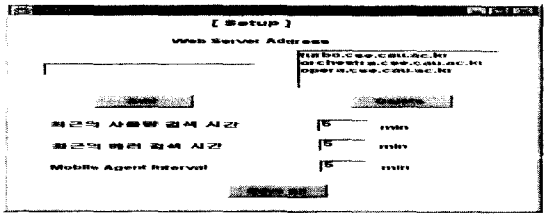


그림 10. 설정 대화 상자
Fig. 10. Setting Dialog box.

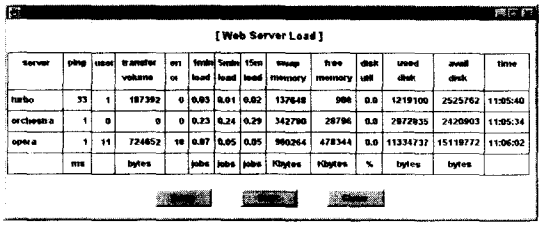


그림 11. 부하 정보 수집 결과
Fig. 11. Result of load informations.

Viewer 대화상자에서 Start 버튼은 선택하면 Slave 에이전트를 일정한 시간간격으로 계속적으로 생성하게 되며 웹서버측에서 전송해온 부하 정보를 사용자에게 보여준다<그림 11>.

Close 버튼을 선택하면 부하 모니터링 작업을 종료하며 Slave 에이전트의 생성도 중단하게 된다.

VI. 성능 평가

이동 에이전트를 이용한 부하 모니터링 시스템의 성능을 평가하기 위하여 클라이언트-서버 패러다임에 속하는 자바 RMI 방법을 사용한 부하 모니터링 프로그램을 구현한 후 두 가지 시스템의 특징과 장단점을 비교·설명하고 웹서버의 메모리 사용량을 측정하였다.

1. RMI를 사용한 부하 모니터링

RMI(Remote Method Invocation)란 자바 언어에서 분산 객체 애플리케이션을 작성하기 위해 제공하는 클라이언트-서버방식이다. RMI를 사용하여 네트워크로 연결된 시스템에 있는 객체들이 서로간의 메소드를 호출하여 원하는 작업을 수행하고 정보를 교환하는 분산 애플리케이션을 작성할 수 있다. RMI 방법을 사용하여 여러 웹서버의 부하를 모니터링하는 프로그램을 작성하기 위해서 웹서버에서 실행될 서버 프로그램과 부하 정보를 수집하기 원하는 클라이언트 프로그램을 작성하였다.

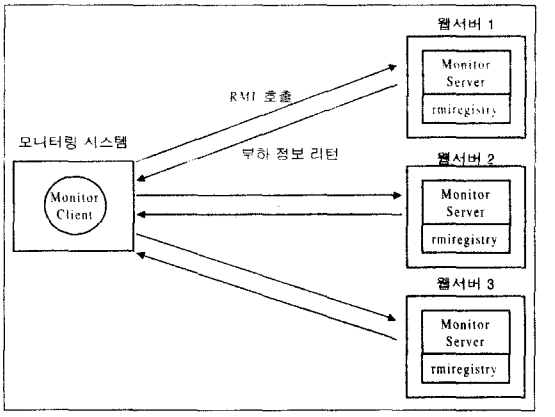


그림 12. RMI를 이용한 부하 모니터링
Fig. 12. Load Monitoring using RMI.

클라이언트 프로그램은 여러 웹서버의 부하 정보를 수집하기 원하는 시스템에서 실행되면서 각 웹서버의 부하 모니터링 작업을 지시하기 위하여 서버 프로그램의 부하 모니터링 작업을 요청한다. 모니터링 작업이 종료된 후 서버로부터 결과값을 리턴받아 그 부하정보를 출력한다. 부하 모니터링 작업은 웹서버의 순서대로 실행하며 일정한 시간간격으로 계속 호출한다. 서버 프로그램은 웹서버에서 계속적으로 실행되면서 클라이언트의 RMI 호출에 대해 부하 모니터링 작업을 수행하는 메소드를 실행한다. 모니터링 작업이 종료되면 그 결과값을 클라이언트에 리턴한다. 서버, 클라이언트 프로그램외에 필요한 부하 모니터링 작업에 필요한 모듈은 앞서 구현한 Monitor 클래스를 그대로 사용하였다. RMI를 사용한 부하 모니터링 프로그램의 실행은 <그림 12>와 같다.

RMI 방법을 사용해서 부하를 모니터링하기 위해서는 먼저 MonitorServer 프로그램과 모니터링 작업을 수행하는 Monitor 프로그램을 각 웹서버로 전송하여 컴파일을 수행하여 클래스 파일을 생성한다. 컴파일한 후, rmic 명령을 사용하여 MonitorServer 클래스의 서터브, 스켈리턴 클래스 파일을 생성해야만 한다. 그리고 클라이언트가 원격 객체의 참조값을 얻기 위해 URL 형태로 요청을 했을 때 서버에 등록된 원격 객체의 이름과 실제 원격 객체를 연결시켜 찾아주는 rmiregistry 프로그램을 실행해야 한다.

각 웹서버에서 네임 서버의 역할을 하는 rmiregistry 프로그램을 실행한 후 MonitorServer 클래스를 실행시키면 서버 프로그램은 계속적으로 실행되면서 클라이언트의 RMI 호출을 받아서 monitoring() 메소드를 실

행한다. 서버의 monitoring() 메소드는 Monitor 클래스에서 구현된 메소드를 사용하여 웹서버의 내부에서 부하를 모니터링하여 그 결과값을 클라이언트에게 리턴한다.

클라이언트 측에서 실행되는 MonitorClient 클래스는 모든 웹서버에 있는 monitoring() 메소드를 차례대로 호출하여 부하정보를 리턴받는, 반복적인 요청-응답의 프로토콜 방식을 사용한다. 클라이언트의 RMI 호출은 일정한 시간간격으로 계속적으로 실행된다.

2. RMI와 이동 에이전트를 이용한 부하 모니터링의 성능 비교

두 가지 방식을 사용한 부하 모니터링 시스템을 구현하여 실험한 환경은 <표 2>와 같다.

RMI를 이용한 부하 모니터링 시스템과 이동 에이전트를 이용한 부하 모니터링 시스템의 특징과 장점을 <표 3>에서 비교하였다.

RMI를 사용한 부하 모니터링 시스템에서는 모든 웹서버마다 서버 프로그램을 설치해야 하지만 이동 에이전트를 사용한 부하 모니터링 시스템에서는 서버 프로그램의 설치가 필요없기 때문에 웹서버 수를 확장하기가 용이하다. RMI에서는 전체 메시지 교환 형태가 중앙집중식인 반면 하나의 이동 에이전트가 여러 웹 서버를 이동하면서 작업을 수행하기 때문에 메시지 전송 형태가 한곳에 집중되지 않고 전체 네트워크상에 분산된 형태를 가지게 된다. 이러한 것은 네트워크 부하를 한쪽 방향에 집중시키지 않고 한쪽의 네크워크 오류에 대비할 수 있는 장점을 가진다.

RMI에서는 스터브, 스켈리턴 클래스를 생성해야 하며, 클라이언트 측으로 스터브 클래스의 동적 로딩이 발생하지만 이동 에이전트에서는 별도의 클래스를 생성할 필요가 없기 때문에 실행 프로그램의 크기가 감소되며, 스터브 클래스의 동적 로딩이 발생하지 않기 때문에 전체 네트워크 사용량이 감소된다.

RMI에서는 서버 프로그램이 모든 웹서버에서 지속적으로 실행되지만 이동 에이전트는 부하 모니터링 작업을 마친 후에 완전히 종료되기 때문에 웹서버에 대한 추가적인 부하 부담이 적다. 웹서버에서 실행되는 클래스의 크기를 살펴보면 RMI가 13.2KB인 반면 이동 에이전트에서는 11.0KB의 크기를 가진다.

실제로 두 가지 방법의 부하 모니터링 시스템을 24시간동안 실행하면서 웹서버 1의 메모리 양을 측정하였다.

표 2. 성능평가의 실험환경
Table 2. Experimental environment of performance evaluation.

	웹서버 1	웹서버 2	웹서버 3	PC
CPU	Pentium Pro 200 MHz	hyperSPARC 125MHz 2개	UltraSPARC 168MHz 4개	Pentium 133MHz
OS	Solaris 2.5.1	Solaris 2.5.1	Solaris 2.5.1	Windows 98
RAM	64MB	128MB	518MB	32MB
언어	JDK 1.1.8	JDK 1.1.8	JDK 1.1.8	JDK 1.1.8
MAS	Aglet 1.1 Beta 1			

표 3. 이동 에이전트와 RMI를 사용한 모니터링 시스템의 성능 비교
Table 3. The performance comparison between mobile agent and RMI.

	이동 에이전트	RMI
서버 프로그램 설치	불필요	필요
웹서버 수의 확장	용이	어려움
메시지 교환 형태	분산 형태	중앙집중식
스터브, 스켈리턴 클래스 생성	불필요	필요
스터브 클래스의 동적 로딩	없음	발생
웹서버에서의 프로그램 실행	실행 후 종료	지속적 실행
네트워크를 통해 전송되는 프로그램 크기	11.0 KB	13.2 KB

부하 모니터링 작업은 5분간격으로 수행하였으며 24시간동안 계속 실행하면서, 웹서버1에서 5분간의 평균 프리 메모리(free memory) 양을 계속 측정하였다. 프리 메모리 양은 사용가능한 가상 메모리(swap memory)와 사용가능한 실제 메모리(real memory)의 양을 더한 값이다. RMI를 사용한 부하 모니터링 시스템과 이동 에이전트를 사용한 부하 모니터링 시스템을 실행하였을 때 웹서버의 프리 메모리 양을 <그림 13>에서 비교하였다.

RMI와 이동 에이전트를 사용한 부하 모니터링 시스템을 24시간동안 실행시키면서 평균 프리 메모리 양을 측정하여 비교한 결과, <그림 13>과 같다. 이동 에이전트를 사용하였을 때의 프리 메모리 양의 평균값은 131,615KB 이며, RMI를 사용하였을 때의 프리 메모리 양의 평균값은 127,033KB였다. 이동 에이전트를 사용한 시스템의 프리 메모리 양이 평균적으로 4,582KB 더

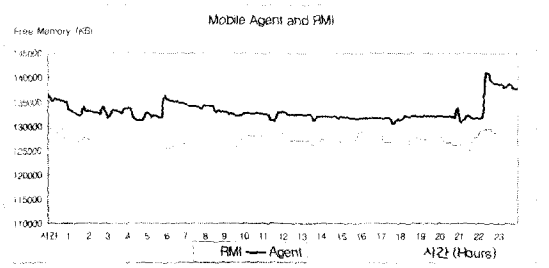


그림 13. 두 가지 시스템의 프리 메모리 양 비교
Fig. 13. The free memory size of two system.

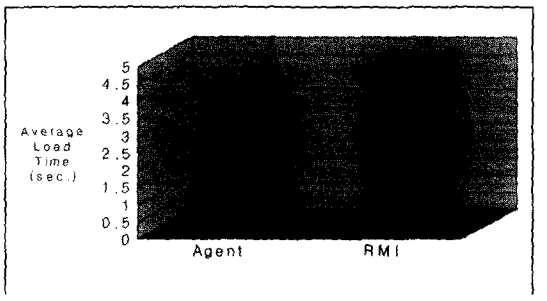


그림 14. 평균 부하 반응 시간
Fig. 14. Average Load Response Time.

많았다. 이러한 현상은 RMI 방법을 사용했을 경우 서버 프로그램이 계속 실행되면서 웹서버의 메모리를 일정량 계속적으로 점유하고 있는 반면에, 이동 에이전트를 사용했을 경우 Slave 에이전트가 모니터링 작업만 수행한 후 완전히 종료되기 때문에 발생한다.

<그림 14>는 본 논문에서 제시하고 있는 부하 정보를 모니터링할 경우 RMI와 이동 에이전트간에 부하 반응 시간을 나타내고 있다. 1,000번 수행후에 RMI와 이동 에이전트간에 평균적으로 소요되는 반응 시간을 의미하는 <그림 14>는 이동 에이전트 경우 평균 3.89초이며, RMI 경우 4.67초를 나타내고 있다. 이동 에이전트 경우 웹 서버로 직접적으로 에이전트가 이주하여 수행되기 때문에, RIM 처럼 각각의 부하 정보에 대한 응답 메시지를 전송할 필요가 없기 때문에 본 논문에서는 이동 에이전트가 RMI보다 약간의 평균 부하 반응 시간이 빠름을 나타내고 있다.

Ⅶ. 결 론

기존의 클라이언트-서버 방식인 RPC, CORBA, Java RMI을 사용한 웹서버 부하 모니터링은 웹 서버 마다 서버 프로그램을 미리 설치해야 하며, 클라이언트 요청 메시지마다 서버측에서 응답 메시지를 전송함으로써 네트워크 트래픽을 증가시키고, 웹서버에 추가적인 부하 부담을 주는 문제점이 있다.

본 논문에서는 클라이언트-서버 방식의 문제점을 해결하기 위해서 이동 에이전트를 이용한 웹서버 부하 모니터링 시스템을 구현하였다. 웹서버 부하 모니터링 작업에 이동 에이전트를 사용함으로서 웹서버에 미리 서버 프로그램을 설치할 필요가 없으며, 네트워크 사용량을 감소시킬 수 있고, 웹서비스에 치중하는 웹서버에 추가적인 부하 부담을 감소시킬 수 있다. 또한 웹서버의 수와 위치에 관계없이 효율적으로 실행 가능하다.

이동 에이전트를 사용한 부하 모니터링의 성능을 RMI 방법을 사용한 부하 모니터링과 비교해보면 서버 프로그램을 설치할 필요가 없기 때문에 웹서버 수의 확장이 쉬우며, 스터브·스켈리턴 클래스를 생성할 필요가 없으며 스터브 클래스의 동적 로딩 현상이 발생하지 않기 때문에 네트워크 사용량을 감소시킬 수 있다. 이동 에이전트는 부하 모니터링 작업이 끝난 후에는 완전히 종료되기 때문에 웹서버에 추가적인 부하 부담을 가하지 않는다.

웹서버에 집중되는 부하를 빠른 시간내에 효율적으로 모니터링 하여 지리적으로 분산된 여러 웹서버의 성능 관리를 위해 이동 에이전트를 이용한 웹서버 부하 모니터링 기술이 여러 웹서버 관리자에 의해 활용될 수 있다.

참 고 문 헌

[1] C. Picoto, P. Veiga, "Management of a WWW Server using SNMP", Proceedings JENC6, 1994.
[2] "Client Side Load Balancing for the Web", Technical Report, http://www.camb.opengroup.org/RI/www/prism/load_bal.html
[3] "Web Server Monitoring", White Paper, http://www.freshtech.com/WhitePaper.html
[4] D. Kotz, R. S. Gray, "Mobile Agents and the Future of the Internet", ACM Operating Systems Review, V.33 N.3, p. 7~13, 1999.
[5] A Perl Script for Monitoring Apache Server Status, http://webreview.com/pub/1999/02/webm/index.html.
[6] Web Site Monitoring, http://www.digitalventures.

net/vp/index.html

[7] D. B. Lange, Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley press, 1998.

[8] D. B. Lange, "Mobile Objects and Mobile Agents : The Future of Distributed Computing?", Proceedings of The European Conference on OOP, 1998.

[9] MARS(Monitoring Application for Resources and Servers), http://www.altara.org/mars.html.

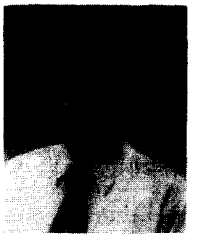
[10] Michael D. Cooper, "Design Considerations in Instrumenting and Monitoring Web-Based Information Retrieval Systems", Journal of the American Society for Information Science, 1998.

[11] M. Caprini, R. Nacach, Z. Qian, "Java Mobile Agent for monitoring task : evaluation report", http://atddoc.cern.ch/Atlas/Notes/078/Note078-1.html, 1998.

[12] E. Anderson, D. Patterson, "Extensible, Scalable Monitoring for Clusters of Computerts", Proceedings of 1997 LISA Conference, 1997.

[13] 박홍진, 정경진, 김영찬, "자바 이동 에이전트를 이용한 웹 서버 부하 모니터링", 한국정보과학회 '99 가을 학술발표논문집(III) 제26권 2호, pp. 489~491, 1999. 10

저 자 소 개



朴 洪 珍(正會員)

1993년 : 원광대학교 컴퓨터공학과 졸업(공학사). 1995년 : 중앙대학교 컴퓨터공학과(공학석사). 2001년 8월 : 중앙대학교 컴퓨터공학과(공학박사). 2001년 9월~현재 : 상지대학교 이공과대학 컴퓨터정보공학부 전임강사. <주관심분야 : 에이전트 시스템, 분산 시스템, 실시간 시스템>