Journal of Korea Multimedia Society Vol. 7, No. 10, October 2004(pp. 1421-1435)

트래픽 감지를 통한 최적 경로 탐색 이동 에이전트 모델 설계 및 구현

김광종[†], 고 현[††], 김영자[†††], 이연식[††††]

요 약

기존 이동 에이전트 모델은 사용자로부터 다양한 작업을 부여 받았을 때, 수동적 라우팅 스케줄 지정에 따라 많은 트래픽이 발생한 시점에서 노드 간을 이주하며 작업을 처리한 후 작업 결과를 서비스하기까지 분산 노드들에 대한 순회 작업 처리 시간 비용이 증가하게 된다. 따라서 기존 이동 에이전트 시스템들은 에이전트 시스템들 간의 에이전트의 크기 증가나 상호 호환성 결여를 해결하기 위한 이주 기법이 요구된다. 이에 본 논문에서는 이동 에이전트의 노드 이주시 네트워크 트래픽 감지를 통한 최적 경로 탐색에 의해 분산된 노드들로의 효율적 작업 처리를 위한 CORBA 기반의 이동 에이전트 모델(CORBA-based Mobile Agent Model : CMAM)을 설계 및 구현한다.

Design and Implementation of Mobile Agent Model Supporting the Optimal Path Search through Traffic Sense

Kwang-jong Kim[†], Ko Hyun[††], YoungJa-Kim[†††], Yon-sik Lee[††††]

ABSTRACT

When various work was given from user, the existing mobile agent model migrated from node to node and process the work at the point that much traffics according to passive routing schedule are happened, until it makes service the work result, time cost of round work process for the distributed nodes are increased. Therefore, the existing agent systems need migration method to solve the problems of agent size increment or a lack of interoperability among agent systems. To solve these problems in this thesis, we design and implement an CORBA-based Mobile Agent Model(CMAM) for the efficient work process into the distributed nodes, when the mobile agents are migrated among the nodes, using an optimal path search by sensing of network traffic.

Key words: Mobile Agent(이동 에이전트), Push Agent(푸시 에이전트), Mobile Agent Facilities(MAF)

1. 서 론

대부분의 기존 이동 에이전트는 분산된 노드들에 대한 라우팅 스케줄(Routing Schedule)이 지정되어 네트워크에 연결된 다수의 에이전트 시스템들로 이주할 경우, 통신망 결손이나 노드 장애에 대한 이주 보장이 제공되지 않음으로써 에이전트가 무한대기 및 고아(Orphan) 상태에 빠지거나 쓰레기(Garbage)로 처리될 수 있다[1]. 또한, 사용자에 의한 수동적인 라우팅 스케줄 지정으로 이주 노드를 재지정 해야

※ 교신저자(Corresponding Author) : 김광종, 주소 : 전북 군산시 미룡동 산68번지(573-701), 전화 : 063)469-4553, FAX : 063)469-4560, E-mail : kkjkim@kunsan.ac.kr
접수일 : 2003년 11월 29일, 완료일 : 2004년 4월 20일
[†] 준회원, 군산대학교 컴퓨터정보과학과 이학박사
[††] 군산대학교 대학원 컴퓨터정보과학과 박사과정 (E-mail : kogo@kunsan.ac.kr)
[†††] 군산대학교 컴퓨터정보과학과 박사수료 (E-mail : tiny89@kopo.or.kr)
[††††] 정회원, 군산대학교 컴퓨터정보과학과 교수 (E-mail : yslee@kusnan.ac.kr)

하는 불가피한 상황이 발생 시 동적으로 이주 노드를 지정할 수 없으므로 이주 시 발생할 수 있는 문제들에 대해 능동적으로 대처할 수 없다. 그러므로 많은 네트워크 트래픽이 발생한 시점에서 노드 간을 이주하며 작업을 처리한 후, 작업 결과를 서비스하기까지 분산 노드들에 대한 순회 작업 처리 시간 비용이 증가하게 된다. 따라서 기존의 이동 에이전트 시스템들이 가진 에이전트의 크기 증가나 에이전트 시스템들 간의 상호 호환성 결여를 해결하기 위한 에이전트 시스템 개발과 수동적인 라우팅 스케줄 지정에 의해 발생할 수 있는 문제들을 해결하기 위한 이주 방안 및 기법이 요구된다.

본 논문에서는 네트워크 트래픽 감지를 통한 동적인 라우팅 스케줄 지정에 따라 이동 에이전트가 분산된 노드들을 자율적으로 이주하여 복잡하고 다양한 작업들을 효율적으로 처리하는 CORBA 기반의 이동 에이전트 모델(CORBA-based Mobile Agent Model : CMAM)을 설계 및 구현한다. 제안된 모델은 이동 에이전트가 이주할 분산된 노드들에 대해 동적으로 라우팅 스케줄을 지정하는 MAFFinder와 에이전트에 대한 수행 패턴 중 Locker 패턴[2]의 적용을 위해 푸시 에이전트를 포함하며, 에이전트 객체 크기에 따른 네트워크 부하를 감소시키기 위해 CORBA의 분산 객체 형태에 적용시켜 에이전트 호출 모듈만을 포함한 이동 에이전트 객체와 작업 실행 모듈을 가진 푸시 에이전트 객체로 각각 분리한다. 그리고 네트워크 트래픽 발생 경로나 시점에서의 에이전트 이주 시 네트워크 트래픽을 감지하여 목적 노드까지의 최적 경로 탐색을 통해 자동적인 경로 조정을 수행하고, 물리적 장애에 능동적으로 대처하여 에이전트의 신뢰성 있는 이주를 지원하도록 하는 이주 기법과 알고리즘을 제시하며, 이주 대상 노드들에 대한 위치 투명성을 보장하는 MAFFinder를 설계 및 구현한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로서 OMG의 MAF 명세와 기존 이동 에이전트 시스템들의 이주 정책에 대해 기술하고, 3장에서는 기존 이주 정책들의 문제점 및 미비점을 보완하여 에이전트의 이주 신뢰성을 보장하고 최소의 네트워크 소요 시간을 갖도록 하는 새로운 에이전트 이주 기법 및 알고리즘과 에이전트의 이주 기법을 통해 보다 효율적으로 작업을 처리하는 이동 에이전트 모델의 구조를 제시한다. 또한, 모델의 구성요소인 MAFFinder, 이동 에이전트, 푸시 에이전트를 설계 및 구현한다. 4장에서는 구현된 이동 에이전트 모델과 기존의 이동 에이전트 시스템을 이용하여 각각 에이전트의 크기와 네트워크 트래픽에 따른 이주 사례에 대한 분석을 통해 제안된 모델의 전체 성능을 평가한다. 마지막 5장에서는 결론 및 향후 연구방향을 제시한다.

2. 관련연구

2.1 상호운용성을 지원하는 OMG MAF

이동 에이전트 기술에서 가장 중요한 것은 다양한 에이전트 시스템 사이의 상호운용성이다. OMG의 MAF(Mobile Agent Facilities) 명세[3]은 에이전트 생성, 이동 그리고 에이전트 시스템과 이동 에이전트의 위치 추적 기능 등 상호운용성을 위한 기본적인 기능들을 지원한다. 이러한 MAF 명세에 기반한 이동 에이전트 시스템은 이동 에이전트 생성, 현재 수행 중인 에이전트의 중지, 중지된 에이전트의 재실행, 에이전트의 종료와 같은 에이전트 관리 기능의 표준화를 통해 다른 타입의 이동 에이전트 시스템을 제어할 수 있다. 또한, 다양한 파라미터 정의나 이동 에이전트 이름, 이동 에이전트 시스템의 이름, 그리고 이러한 객체들의 위치에 대한 정보 등을 표준화함으로써 에이전트 시스템에 이주하려는 에이전트의 적합성을 결정할 수 있고, 이름을 통한 상대 이동 에이전트를 인식할 수 있다. 그림 1은 이종의 에이전트 시스템 간의 상호운용성을 지원하기 위한 MAF 기반의 이동 에이전트 시스템이다.

MAF 명세는 이동 에이전트 생성 및 목적 에이전트 시스템으로의 에이전트 이주를 지원하는 MAFAgentSystem 인터페이스와 네이밍 서비스(Naming Service)를 제공하는 MAFFinder 인터페이스로 구성된다[3]. MAFAgentSystem 인터페이스는 이동 에이전트의 생성 및 수행 중인 에이전트의 중지, 중지된 에이전트의 재실행, 에이전트 종료 등 에이전트 관리 기능을 지원하는 메소드와 객체를 정의하고 있다. 이 인터페이스의 메소드들은 이동 에이전트 시스템이 에이전트를 전송하는데 갖추어야할 기본적인 오퍼레이션으로써 상호운용성을 지원하기 위한 것이다[3,4].

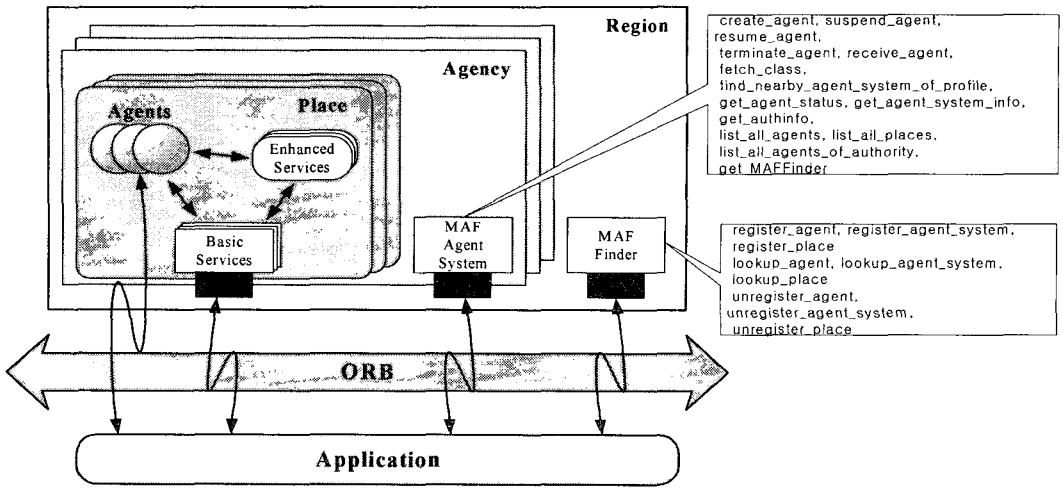


그림 1. 상호운용성을 위한 이동 에이전트 시스템

2.2 기존 이동 에이전트의 이주 정책

대부분의 이주 보장 정책들은 이동 에이전트가 통신망에 연결된 다수의 이동 에이전트 시스템들로 이주할 때 발생할 수 있는 통신망 혹은 노드 결점에 대처할 수 있는 이주 정책[5-10]들이다. 통신망 결손이나 노드 장애 또는 데이터베이스 등 정보 서비스의 부재 시 이동 에이전트는 무한 대기나 고아 상태에 빠질 수 있고 혹은 파괴되어 쓰레기로 처리될 수 있다[1,11-15].

따라서 이동 에이전트의 이주 시 발생할 수 있는 통신망 결손 및 노드 장애에 대처하기 위해, JAMAS [13]에서는 경로 재조정과 후위 복구 기법을 제안하였고, [5,9,13]에서는 방문하는 노드의 결점에 대처하기 위한 프로토콜을 제시하였다. 또한, Mole[6-8]은 이동 에이전트에 대한 고아 찾기와 성공적인 종료를 위해 그림자(Shadow) 프로토콜을 제공하고 있으나, 이 프로토콜은 이동 에이전트가 이주 중에 결점이 발생한 경우에만 이를 찾아 처리할 뿐 에이전트의 이주 신뢰성은 보장하지 않는다. 무결점(Fault-Tolerance)을 지원하는 Mole[6-8]은 전체 노드에 이동 에이전트를 복사함으로써 투표(Voting)와 선택(Selection) 프로토콜을 이용한 효율적인 이주 기법을 제공하지만, 작업(Worker) 노드를 제외한 모든 관찰(Observer) 노드 간의 상호 감시와 통신 기능이 필요하며, 통신망 접속에 결점이 없는 것을 가정하고 있다.

이와 같이 이동 에이전트의 이주 전략과 관련하여 통신망 혹은 노드의 결점이 발생하였을 때 이주 신뢰성을 보장하기 위한 연구가 주류를 이루고 있으며, 최소의 네트워크 소요시간을 갖도록 이동 에이전트의 이주 계획을 세우는 알고리즘에 대한 연구는 아직 미흡하다. 따라서 효율적인 이동 에이전트 시스템의 개발을 위해서는 이동 에이전트를 어떻게 구성할 것인가와 어떠한 이주 방법을 통해 에이전트를 이주시킬 것인가가 매우 중요하다.

3. 이동 에이전트의 노드 간 이주 기법

3.1 노드 이주 시간 최소화 방법

기존 이동 에이전트는 사용자로부터 복잡하고 어려운 문제를 부여받았을 경우, 작업 처리를 위한 실행 로직의 크기가 커져 전체 에이전트 객체의 크기를 증가시킬 수 있다. 또한, 에이전트의 작업 처리 결과가 매우 많은 경우에도 에이전트 객체의 크기를 증가시켜 네트워크의 과부하를 발생시킬 수 있다. 따라서 이러한 에이전트의 크기에 따른 네트워크 과부하는 이동 에이전트의 노드 이주 시간에 많은 영향을 미친다. 그러므로 기존 이동 에이전트 시스템들이 가진 문제들을 해결하기 위해 두 가지 방법을 사용한다. 먼저, 이동 에이전트의 크기에 따른 네트워크 부하 문제는 에이전트 작업 실행 모듈의 분리 및 Locker 패턴[2]을 응용한 푸시 에이전트의 개발을 통해 해결하고, 네트워크 트래픽으로 인한 이동 에이전트의 이주 지연 문제는 트래픽 감지를 통한 최적의 이주 경로 탐색 및 조정 기법을 적용하여 이동 에이전트가

최적의 이주 경로로 이주를 수행하도록 한다.

네트워크 과부하 문제를 해결하기 위한 이동 에이전트는 CORBA의 분산 객체 형태를 적용하여 작업 실행 모듈을 가진 CORBA 구현 객체는 푸시 에이전트 객체로, 호출 모듈을 가진 CORBA 클라이언트 객체는 이동성 객체인 이동 에이전트 객체로 구현함으로써 두 에이전트 객체 간의 상호 협력을 통해 작업을 처리할 수 있도록 구성한다. 또한, 에이전트가 처리된 결과를 가지고 다음 호스트로 이주하지 않고 대신 푸시 에이전트 객체가 처리된 결과를 사용자에게 직접 전달하도록 하여 실행 시의 에이전트 크기를 일정하게 유지한다. 이러한 방법으로 구성된 이동 에이전트의 수행 패턴은 그림 2와 같다.

3.2 네트워크 트래픽 감지 방법

네트워크 트래픽 측정에 있어 원격 호스트에 하나의 패킷을 보내 패킷의 왕복 시간을 검사하기 위해서는 Ping 프로그램을 사용한다. Ping은 TCP/IP 프로토콜을 사용하는 응용 프로그램으로 다른 호스트에 IP 데이터그램이 도착할 수 있는지를 검사한다. Ping을 수행하는 프로그램은 ICMP(Internet Control Message Protocol) Echo Request라는 메시지를 원격 호스트로 보내 응답하는 지의 여부에 따라 원격 호스트가 동작 중인지 아닌지를 검사할 수 있다. 그러나 이러한 Ping은 가공되지 않은 IP 데이터를 전송하는 ICMP를 사용함으로 이를 지원하지 않는 JAVA로는 구현할 수 없다. 따라서 Ping 프로그램의 실행 정보를 JAVA 어플리케이션에서 이용하려할 경우, 그림 3의 알고리즘에서와 같이 Runtime 클래스의 exec() 메소드를 사용하여 프로세스를 생성한 후 InputStream 클래스를 통해 Ping 프로그램의 실행 정보를 획득하여 이를 이용한다.

3.3 최적 이주 경로 탐색 및 조정 기법

이동 에이전트 시스템에서 사용자 요구사항에 대한 작업 처리 시간은 에이전트가 부여받은 임무를 수행하는 비즈니스 로직의 실행 시간과 분산된 노드들 간을 순회하는데 걸리는 에이전트 이주 시간이 포함된다. 최적 이주 경로 탐색 및 조정 방법은 이러한 이동 에이전트의 이주 시간을 단축시키기 위한 방법으로, 에이전트가 이주할 노드들 중 최소의 네트워크 소요 시간을 갖는 경로를 선택하는 것이다. 즉, 네트워크의 부하 발생 요인인 트래픽 감지를 통해 과다한 트래픽이 발생하는 경로는 회피하고 최소의 네트워크 지연을 갖는 경로를 선택하여 보다 효율적인 작업 처리가 이루어지도록 하기 위한 방식이다.

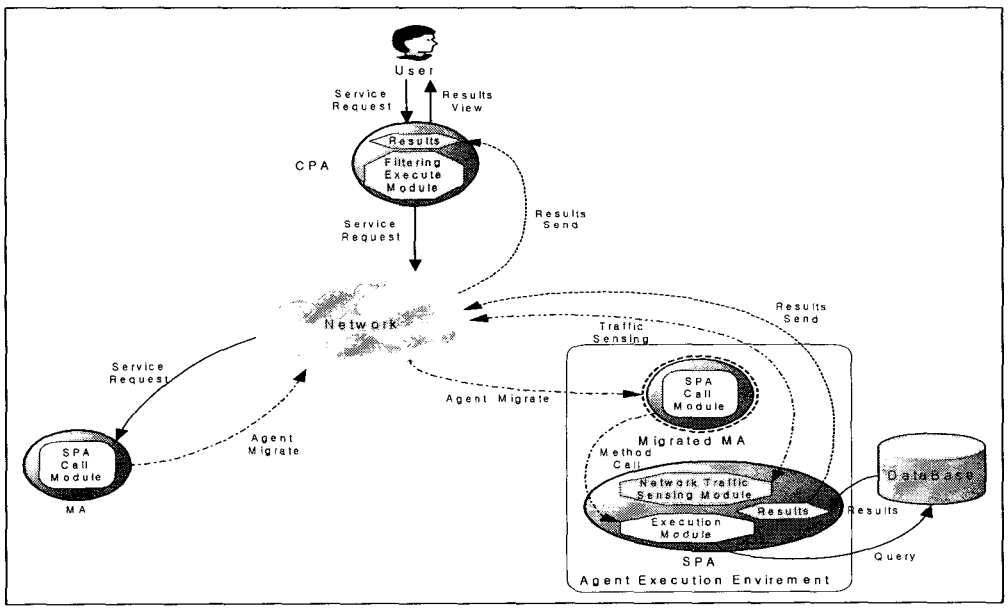


그림 2. 이동 에이전트의 수행 패턴

```
public class Ping {
    public static void main(String[] args) {
        try {
            // Windows 운영체제의 Ping 명령을 실행시키는 Process를 생성한다.
            Process p = Runtime.getRuntime().exec("ping -n 10 202.31.147.142");
            byte[] msg = new byte[500];
            int len;

            // Process에 의해 실행된 Ping 정보를 InputStream을 통해 msg크기의
            // byte 만큼 읽어들여 출력한다.
            while((len=p.getInputStream().read(msg)) > 0) {
                System.out.print(new String(msg, 0, len));
            }
                    .....
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

그림 3. JAVA를 이용한 Ping 프로그램 실행 알고리즘

Ping을 통해 얻어진 값들의 비교 수행을 통해 최상위의 SPA 객체 참조자에 해당하는 노드 경로가 최적이지만 다른 노드 경로들의 순서가 바뀌어 있어도 라우팅 테이블을 재구성하지 않는다. 이는 에이전트가 분산된 노드들을 이주하며 작업을 수행함으로써 이주한 노드들의 위치에 따라 이들의 순서가 새롭게 바뀔 수 있기 때문이다. 한편, 최적 경로 조정 과정은 최상위의 SPA 객체 참조자에 해당하는 노드 경로가 최적이 아닌 경우에만 패킷의 평균 왕복 시간과 패킷 분실 수의 값이 최소인 노드 경로들 순으로 라우팅 테이블을 재구성한다. 이 때, 패킷의 평균 왕복 시간 및 분실된 패킷의 수를 이용한 값 비교 시 패킷 분실 수가 적은 것을 우선으로 라우팅 테이블을 재구성하게 되는데, 이는 통신망의 결손 발생으로 인한 이동 에이전트의 이주 보장을 위해서이다. 예를 들면, 두 노드에 대한 A와 B경로의 평균 패킷 왕복 시간이 각각 10m/s와 15m/s이고 패킷 분실 수가 각각 5와 3일 경우, B경로의 노드를 선택하여 에이전트 이주를 수행한다. 그림 4는 이러한 최적의 이주 경로 탐색 및 경로 조정 방식을 수행하는 과정이며, 그림 5는 이러한 과정을 구현한 최적 이주 경로 탐색 및 조정 기법의 알고리즘이다.

3.4 노드 장애에 따른 이주 노드 재지정 기법

이동 에이전트가 통신망의 결손으로 인해 이주하지 못하는 경우에는 3.3절에서 설명한 바와 같이

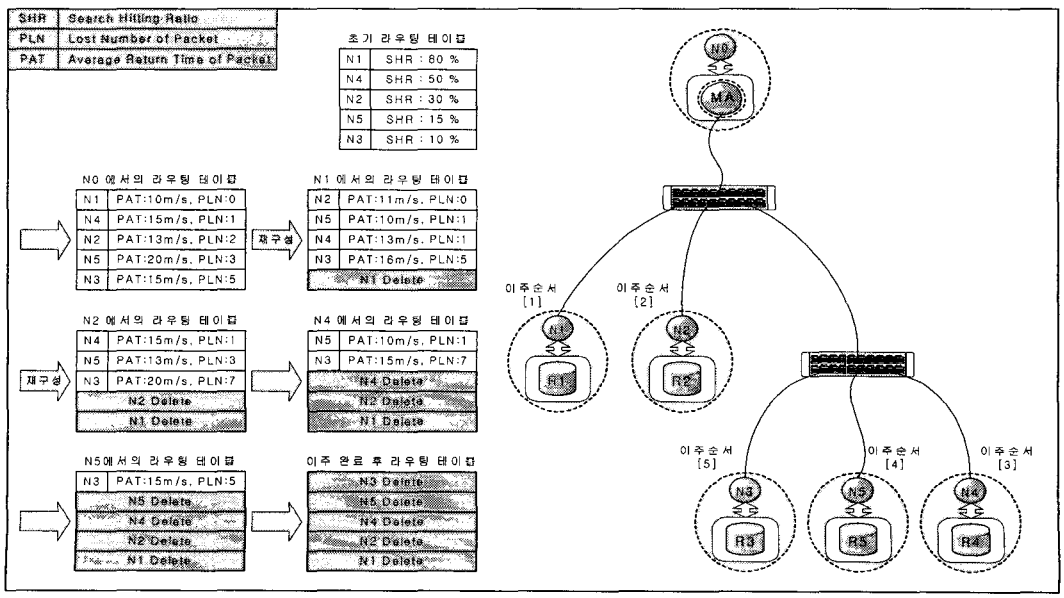


그림 4. 최적 이주 경로 탐색 및 조정 방식의 수행 과정

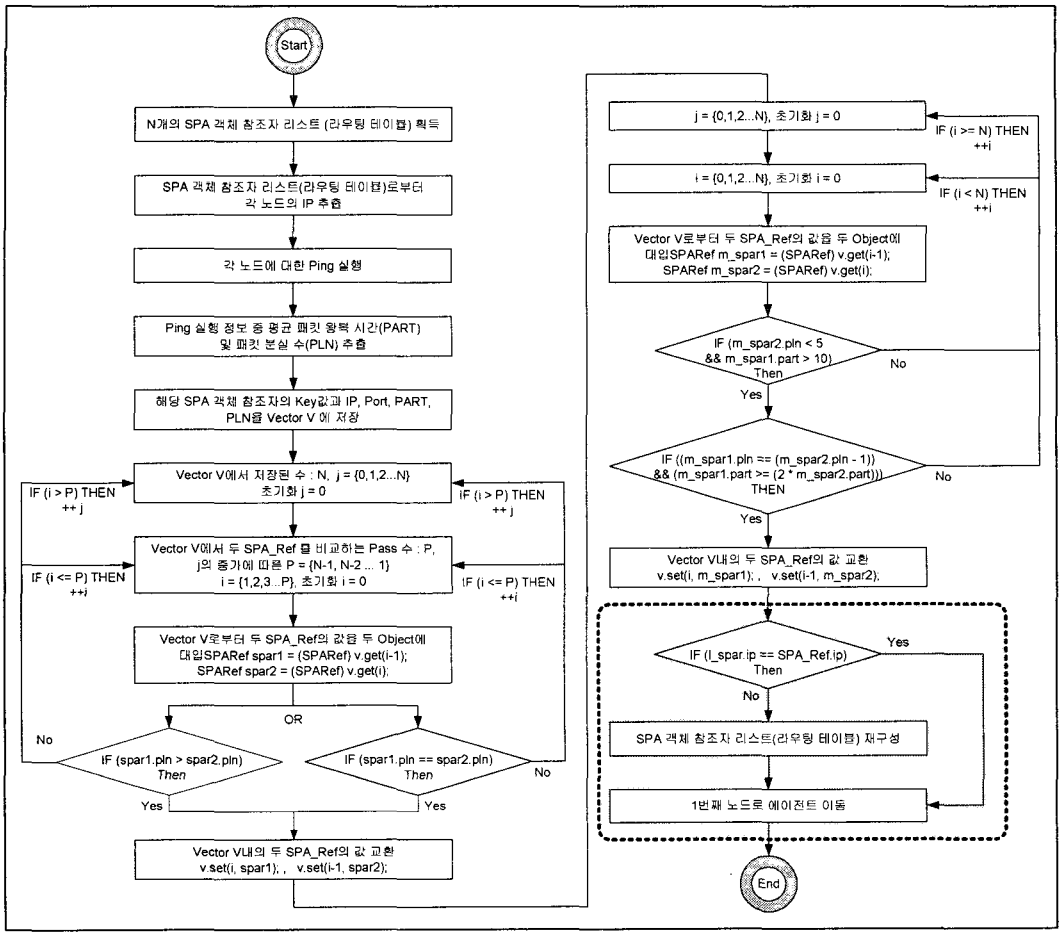


그림 5. 최적 이주 경로 탐색 및 조정 알고리즘

Ping의 실행 정보 중 분실된 패킷의 수인 Lost 값을 추출하여 이 값이 최소인 경로로만 에이전트의 이주를 수행함으로써 이주 신뢰성을 보장한다. 한편, 노드의 장애로 인해 이미 이주하여 실행중인 에이전트가 무한 대기 상태에 빠지거나 쓰레기로 처리되는 경우에는 노드 재지정 방법을 사용하여 해결할 수 있다. 노드 재지정 방법은 장애가 발생한 노드의 이전 이주 노드에서 새로운 이주 노드를 재지정하여 에이전트를 이주시키는 방식이다. 노드 재지정 방식의 수행 과정은 다음과 같다.

① 이동 에이전트는 이미 이주하여 작업을 수행한 A 노드로부터 라우팅 테이블에서 최적 이주 노드인 B 노드로 이주한다. 이 때, B 노드로 이주하는 이동 에이전트는 자신을 복제하여 A 노드에 남겨둔다.

② A 노드에 복제되어 남겨진 이동 에이전트는 B 노드로부터 복제 에이전트 삭제 신호가 수신될 때까지 일정한 시간동안 대기한다.

③ 만약, B 노드에 장애가 발생하여 A 노드로부터 이주한 에이전트가 고아 상태에 빠지거나 파괴되었을 경우, A 노드의 복제 에이전트는 정해진 대기 시간이 지나면 B 노드에서 장애가 발생했다고 판단하고 라우팅 테이블의 우선 순위에 따라 B 노드가 아닌 다음의 최적 노드인 C 노드로 에이전트를 재이주시킨다.

④ B 노드에서 장애가 발생하지 않고 이주된 이동 에이전트가 주어진 작업을 완전하게 처리하면, B 노드의 이동 에이전트는 다음 노드로 이주하기 전 A 노드에 남겨놓은 복제 에이전트를 삭제하도록 신호를 보낸다.

⑤ A 노드는 B 노드의 이동 에이전트로부터 복제

에이전트의 삭제 신호를 수신하고 남겨진 에이전트 객체를 삭제한다.

노드 재지정 방법은 이러한 과정을 통해 이주 노드들의 장애에 능동적으로 대처함으로써 보다 신뢰할 수 있도록 이동 에이전트의 이주를 보장한다. 그림 6은 노드 장애에 능동적으로 대처하는 노드 재지정 방식의 수행 과정이며, 그림 7은 이러한 과정을 수행하는 노드 재지정 알고리즘이다.

3.5 이동 에이전트 모델

3.5.1 이동 에이전트 모델 구조

제안된 모델은 확장된 형태의 MAFFinder를 이용하여 능동적으로 라우팅 스케줄을 지정함으로써 많은 네트워크 트래픽 발생 시 이동 에이전트가 과다한 트래픽 발생 경로를 회피하여 최적의 경로를 통해 이주할 수 있도록 지원한다. 또한, CORBA 기반의 분산 객체 형태로 이동 에이전트를 구현하고, 에이전트의 수행들 패턴 중 Locker 패턴[2]을 적용시켜 에이전트 프레임웍을 구성함으로써 네트워크의 부하를 최소화하여 효율적인 에이전트의 이주가 이루어질 수 있도록 지원한다.

이동 에이전트 모델은 그림 8에서와 같이 사용자의 요구를 받고 이동 에이전트에 의해 처리된 결과를 서비스해주는 사용자 브라우저(User Browser)와 이동 에이전트의 이주 노드에 대한 위치 투명성 및

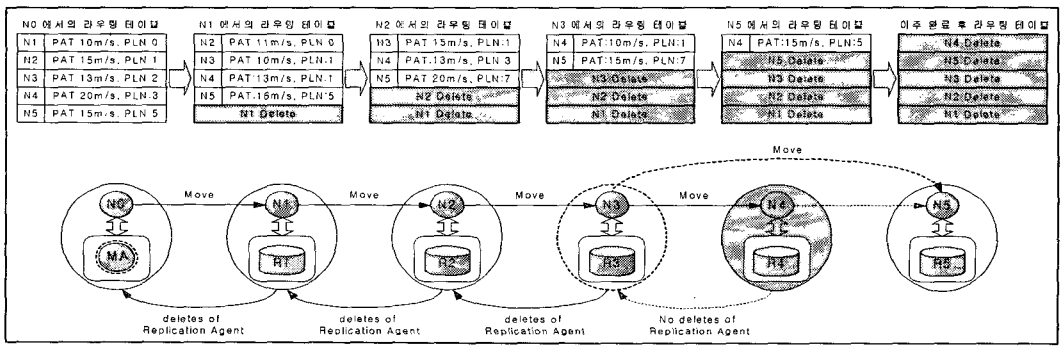


그림 6. 노드 재지정 방식의 수행 과정

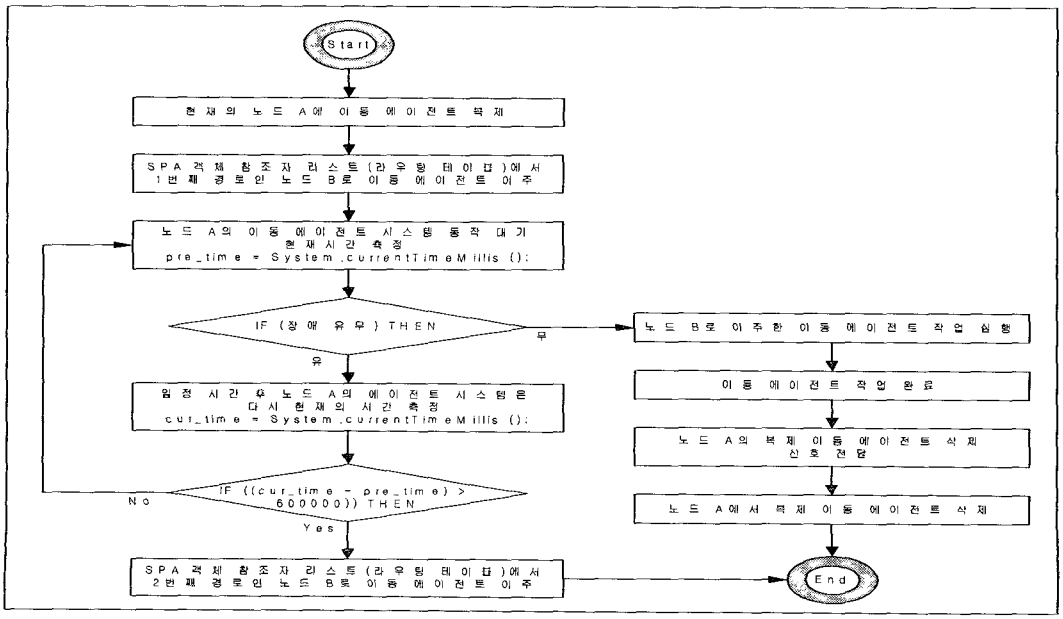


그림 7. 노드 장애에 따른 노드 재지정 알고리즘

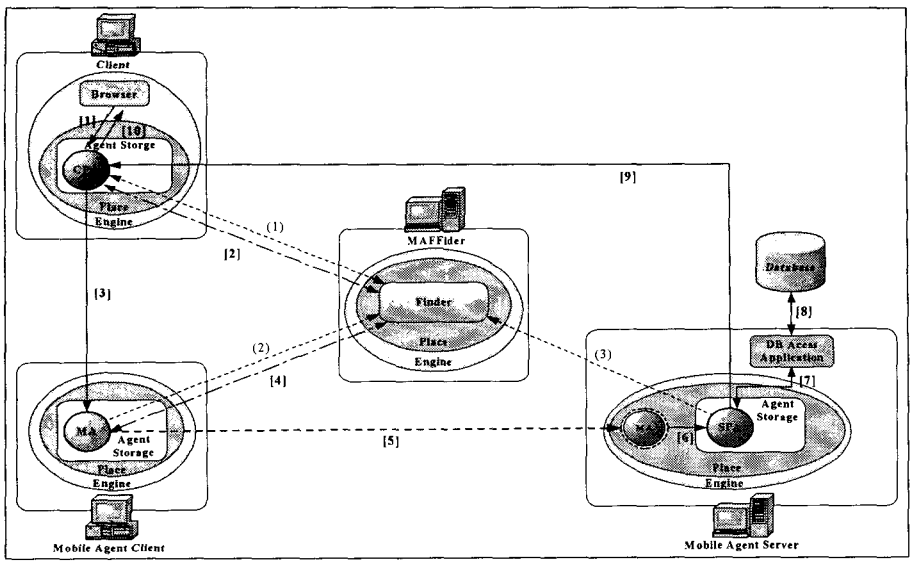


그림 8. 최적 경로 탐색 이동 에이전트 모델 구조

초기 라우팅 스케줄을 지정하는 MAFFinder, 사용자 요구에 대한 이동 에이전트 객체 생성 및 원본 에이전트 객체를 관리하는 이동 에이전트 클라이언트(Mobile Agent Client), 서버 푸시 에이전트를 포함하는 이동 에이전트 서버(Mobile Agent Server)로 구성된다. 그림 8은 능동적인 최적 경로 탐색을 통해 에이전트의 작업을 보다 효율적으로 처리하는 최적 경로 탐색 이동 에이전트 모델이다.

3.5.2 이동 에이전트 모델의 통신 과정

이동 에이전트 모델에서의 통신 과정은 각 에이전트에 관련된 정보 등록 과정과 각 에이전트 간 작업 처리 과정으로 구분된다. 앞서, 3.5.1 절에서 보인 그림 8의 (1)~(3) 과정은 각 에이전트 관련 정보 등록 과정으로, 이동 에이전트 모델을 구성하는 각 시스템들의 초기 기동 시 각각의 시스템 내에 포함된 에이전트, 플레이스, 에이전트 시스템에 대한 객체들을 네이밍 서비스를 지원하는 MAFFinder에 등록하는 과정이다. 등록된 객체 정보들은 에이전트들 간 통신이나 이동 에이전트의 이주 시 원격 시스템들 간의 연결이 가능하도록 위치 투명성을 제공하며, 이동 에이전트가 분산된 각 노드들을 이주하며 작업을 처리할 수 있도록 지원한다. 따라서 사용자의 요구에 대한 이동 에이전트 모델에서의 작업 처리 흐름으로 그 과정은 그림 9와 같다.

3.5.3 MAFFinder

MAFFinder는 이동 에이전트의 이주 노드의 위치 투명성 및 초기 라우팅 스케줄을 지정하기 위해 각 에이전트 시스템 관련 객체들과 사용자가 요구하는 정보 검색 키워드를 관리하는 데몬(Deamon) 객체이다. MAFFinder는 그림 10과 같이 각 객체 정보들을 관리하기 위해 네이밍 서비스별로 스레드를 할당하여 각 스레드별 네임 스페이스를 생성한 후 네임 스페이스 내의 메타 데이터 테이블에 이동 및 푸시 에이전트, 플레이스, 이동 에이전트 시스템 등의 객체를 등록한다. 또한, 정보 검색에 사용되는 키워드들을 저장하기 위해 별도의 네임 스페이스를 생성한 후 등록된 각 에이전트 이름과 객체 참조자, 계층적 검색 키워드 등의 정보를 저장 및 관리한다. 그림 10은 에이전트 관련 객체 참조자와 정보 검색의 정확성을 위해 검색 키워드를 메타 테이블에 관리하는 MAFFinder의 구조이다.

MAFFinder는 에이전트 객체의 이름을 통한 객체 참조자 획득은 물론 사용자에 의해 요청된 검색 키워드만으로도 에이전트의 객체 참조자를 얻을 수 있도록 구현하여 보다 효율적이고 정확한 정보 검색 서비스를 지원하도록 한다. 이를 통해 MAFFinder는 불필요한 노드 이주 방지 및 순회 검색 수행 시간을 단축시킬 수 있는 노드 이주 방법을 제공할 수 있으며, 한정된 네트워크 환경에서의 트래픽을 감소시키

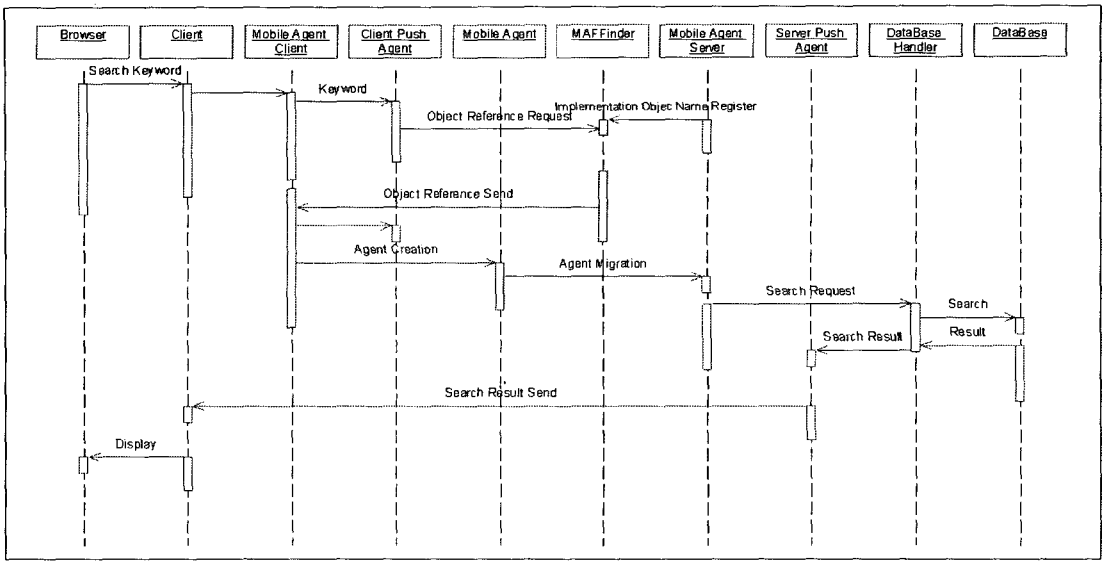


그림 9. 이동 에이전트 모델의 통신과정

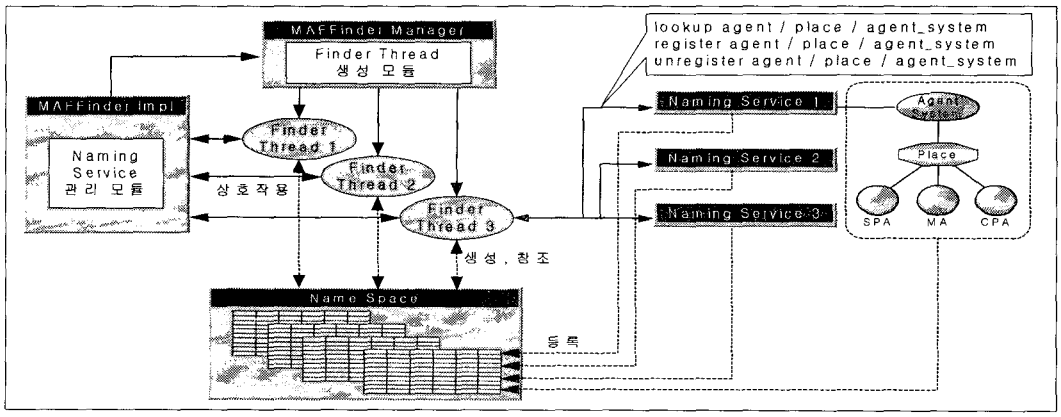


그림 10. MAFFinder 구조

고 다양한 사용자의 요구에 따른 정보 검색의 정확도를 높여 정보 서비스의 질을 향상시킬 수 있다.

MAFFinder는 이동 에이전트와 플레이스, 그리고 이동 에이전트 시스템의 동적인 이름과 위치 정보를 유지하기 위한 lookup_agent(), lookup_place(), lookup_agent_system() 메소드를 제공한다. 이 세 개의 인터페이스를 통해 MAFFinder에 등록된 에이전트와 플레이스, 이동 에이전트 시스템의 위치를 찾을 수 있으며, 이 중 lookup_ agent()는 특정 에이전트 프로파일로 에이전트를 검색할 수도 있다. 또한, MAFFinder는 이동 에이전트 모델을 구성하는 각 시스템 구동 시 시스템 내의 에이전트와 플레이스, 이동 에이전트 시스템을 등록하는 메소드(register_agent, register_place, register_agent_system)를 가지며, 등록된 객체들을 제거하기 위한 메소드(unregister_agent, unregister_place, unregister_agent_system)도 포함한다. 이러한 메소드들은 MAFFinder 구현 시 반드시 구현해주어야 한다. MAFFinder는 이동 에이전트 모델내 각 시스템의 에이전트 등록 및 에이전트 객체 참조 요구에 대해 객체 참조자를 반환하는 기능을 수행한다. 에이전트 등록 및 참조 시 MAFinder의 MAFFindder Manager는 스레드를 생성하여 각 요구에 대한 등록 및 객체 참조자 검색 작업을 실행하고, MAFFinder Impl에서 네이밍 서비스의 관리를

수행한다.

3.5.4 이동 에이전트

이동 에이전트는 사용자로부터 부여받은 작업을 실행하고 다음 대상 목적 노드로 이주하는 이동성 객체이다[1,8,17,18]. 이동 에이전트 시스템의 Agent Generator에 의해 복제된 이동 에이전트는 목적 노드로의 이주를 위해 MAFFinder에 등록된 SPA들 중 사용자의 요구를 처리할 수 있는 SPA에 대한 객체 참조자 리스트를 획득하여 이에 따라 노드 간을 이주하게 된다. 이 때, 최적의 이주 경로를 탐색하기 위해 SPA 객체 참조자 리스트에 포함된 각 노드까지의 경로에 대한 네트워크 트래픽을 검사하여 최소의 네트워크 지연시간을 갖는 경로를 선택한다. 만약, 선택된 경로로 이주한 이동 에이전트가 노드 장애로 인해 무한대기에 빠졌거나 파괴되었을 경우, 원래의 이동 에이전트 시스템은 노드 재지정 모듈을 사용하여 장애가 발생한 노드의 다음 우선 순위 노드로 에이전트를 이주시킨다.

스레드 형태로 동작하는 이동 에이전트는 그림 11의 알고리즘에서 에이전트의 정보를 반환하는 getAgentProfile()과 에이전트를 실행하는 run(), 다음 이주 노드로 이주를 수행하는 gotoNext(), 그리고 이동에이전트가 현 에이전트 시스템에 도착했을 때 수행하는 onArrive()와 이동 에이전트가 현 에이전트 시스템을 떠나기 바로 전에 수행하는 onLeave() 인터페이스를 기본적으로 가진다. 이동 에이전트는 다음 노드로 이주를 수행하기 위해, getOptimalPath() 메소드를 호출하여 최적의 이주 경로를 탐색한 후 선택된 이주 경로를 통해 에이전트를 직렬화하여 이주시킨다. 또한, 이미 이주한 이동 에이전트가 노드 장애로 인해 작업을 처리할 수 없을 때, 이동 에이전트 시스템은 이동 에이전트의 getNodeRedesign() 메소드를 호출하여 장애 노드 다음의 우선 순위에 있는 노드를 선택하여 이동 에이전트가 재이주할 수 있도록 지원하다. 그림 11은 최적 경로를 탐색하여 에이전트를 이주시키는 이동 에이전트의 알고리즘이다.

3.5.5 푸시 에이전트

푸시 에이전트는 기존의 푸시 에이전트가 클라이언트에 지속적인 정보 서비스를 제공함으로써 네트워크 트래픽을 가중시키는 문제를 해결하기 위해서, 네트워크 트래픽 감지를 이용하여 데이터 전송을 지연시키는 방식을 사용한다. 즉, 네트워크 트래픽 감지를 통해 특정 시간에 네트워크 트래픽이 증가했을 경우 실시간적 정보 서비스를 지연시키고, 트래픽이 감소했을 경우 이를 사용자에 서비스하도록 함으로써, 한정된 대역폭의 네트워크 환경에서 안정적이고 신뢰할 수 있는 정보 서비스를 제공할 수 있도록 한다. 이러한 푸시 에이전트의 구현은 CORBA 이벤트 서비스의 객체 간 서비스 방식에 기반하여 에이전트를 생성하고, 에이전트에 네트워크 트래픽 감지 모듈과 서비스 지연 모듈을 추가하여 최적의 데이터 전송이 이루어지도록 한다.

서버 푸시 에이전트는 이동 에이전트로부터 사용자의 요구사항인 정보 검색 키워드와 CPA 객체 참조자를 전달받는다. 서버 푸시 에이전트는 각각 전달받은 검색 키워드와 CPA 객체 참조자를 이용해 데이터베이스를 검색하고, 네트워크 트래픽을 감지하기 위해 Ping 프로그램을 실행시켜 클라이언트 푸시 에이전트에 대한 노드 경로 간 패킷의 왕복 시간과 패킷의 분실 수를 추출한다. 만약, 추출된 패킷의 왕복 시간이 10000m/s 보다 크거나 패킷의 분실 수가 5개 이상이면 검색 결과의 전송을 30000m/s 간 지연시킨다. 검색 결과의 전송 지연 시간이 지나면 서버 푸시 에이전트는 다시 클라이언트 푸시 에이전트에 대한 노드 경로 간 Ping 실행 정보를 검사하여 검색 결과를 전송한다. 한편, 클라이언트 푸시 에이전트는 서버 푸시 에이전트로부터 검색 결과가 전송되면 이를 감지하여 사용자 브라우저에 디스플레이한다. 그림 12는 네트워크 트래픽 감지를 통해 실시간적 정보 서비스를 지연시키는 푸시 에이전트의 알고리즘이다.

4. 실험 및 성능 평가

본 장에서는 기존 이동 에이전트 시스템인 Aglets 시스템과 3장에서 설계 및 구현한 최적 경로 탐색 이동 에이전트 모델을 기반으로 분산 정보 검색 시스템에서 이주 방식에 따른 이주 시간을 평가하기 위한 것이다. 제안된 이동 에이전트 모델은 최적 이주 경로 탐색 및 조정 기법과 이주 노드 재지정 기법을 사용한 모델로서, 이주 노드 재지정 기법은 에이전트 객체의 안정적인 이주를 보장하기 위한 정책이기 때문에 도식 형태로 이에 대한 결과를 표현하기가 어렵

```
public class Agent implement CMASAgent
{
   // Agent 생성자
   public Agent (String name, int age, LinkedList path) {
       agentName = name;
       agentAge = age;
       agentPath = path;
   }

   // 이동 에이전트의 정보 반환
   public AgentProfile getAgentProfile() {
   }

   // 이동 에이전트가 대상 에이전트 시스템에 도착했을 때 수행하는 기능
   public void onArrive() {
   }

   // 이동 에이전트가 현 이동 에이전트 시스템을 떠나기 전에 수행하는 기능
   public void onLeave() {
       // 이주해오기 이전 노드에 복제한 이동 에이전트 삭제
       AgentDelete("jacorba_agent");
                   .. .. ..
   }
```

```
   // 이동 에이전트 실행
   public void run () {
       String s;

       for (int i=0;i<10;i++) {     this.agentAge++;  }

       // SPA 객체참조자 리스트의 크기가 0인지 체크
       if (agentPath.size() != 0) {
          // SPA 객체 참조자 리스트를 최적 경로 탐색 메세지에 넣어줌
          l_spa = (SPARef) getOptimalPath(agentPaht).get(0);
          s = l_spa.host;
          gotoNextHost(s);
       } else {
                   .. .. ..
       }
   }

   // 다음 최적 이주 경로로 에이전트 이주
   public void gotoNextHost (String nextHost) {
       // 다음 노드의 위치를 얻어온다
       dest_mafAgentSystem = mafAgentSystem.getObjectReference(nextHost);
       // Agent 객체 직렬화
       agent = mafAgentSystem.objectToByte(this);
       // Agent Moving
       dest_mafAgentSystem.receive_agent ( ····);
   }

   // 최적 노드 이주 경로 탐색 및 조정 수행
   public Vector getOptimalPath(Vector v) {
   }

   //  재이주를 위한 노드 재조정 수행
   public Vector getNodeRedesign(Vector v) {
   }
                   .. .. ..
}
```

그림 11. 이동 에이전트 알고리즘

다. 따라서, 본 실험에서는 성능 평가를 위해 최적 이주 경로 탐색 및 조정 기법만을 적용한 이동 에이전트 모델을 성능 분석 모델로 정의하여 기존 이주 기법과의 비교를 통해 각 노드로의 이주 시간을 분석 및 평가한다. 실험 환경으로는 서로 다른 이동 에이전트 시스템들 간의 상호운용성 지원을 위해 OMG의 MAF 명세와 CORBA 3.0 스펙(Spec)에 기반한 OrbixWeb 3.2를 사용하였고, 제안된 모델의 구현을 위해 프로그램 개발 도구인 JBuilder 4.0과 JDK 1.4.0을 사용하였다. 또한, Aglets[18-20] 이동 에이전트 시스템과 제안된 이동 에이전트 모델에서의 에이전트 이주 시간 측정 실험은 전체 네트워크의 대역폭이 100 Mbps이고, 노드 간 거리가 15m~20m인 네트워크 환경하에서 수행하였다. 구현된 각 시스템의 운영

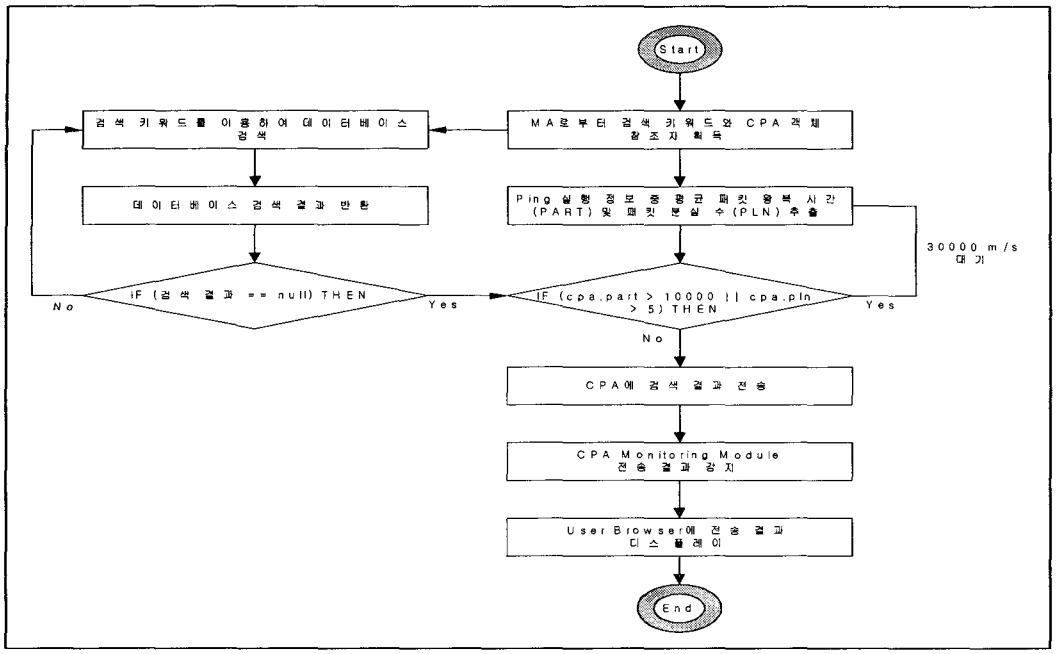


그림 12. 푸시 에이전트 알고리즘

환경은 표 1과 같다.

이동 에이전트의 순회 작업 처리 시간(Mobile agent Round Work Processing Time:MRWPT)은 에이전트 이주 시간(Agent Migration Time:AMT)과 작업 처리 시간(Work Processing Time:WPT)으로 구분된다. 그러나 본 이주 시간 측정 모델은 순수 에이전트 이주 시간인 AMT를 측정하기 위한 실험 모델이지만, 실제 순수 에이전트 이주 시간을 측정하는 자체가 실험 환경적인 여건상 불가능함으로 에이전트 객체가 recevie_agent() 메소드를 호출하는 시점을 전후로 시간을 검사하여 전후 시간 값의 차를 구해 에이전트 이주 시간으로 표현하며 에이전트 크기에 따른 이주 시간 분석은 Aglets 이동 에이전트 시스템과 제안된 이동 에이전트 모델에서 고정적인 라우팅 스케줄에 따라 각 노드로 이동 에이전트를 이주시킬 때, 에이전트의 크기 변화에 따른 이주 시간의 증가량을 측정한 후 이를 비교하여 두 시스템 간의 성능을 평가한다. 본 실험에서는 두 에이전트 시스템의 에이전트 이주 기법에 따른 이주 시간의 차이를 최소화하기 위해, 동일한 에이전트의 라우팅 스케줄에 따라 이주 시간을 측정한다.

4.1 에이전트 크기에 따른 이주 시간 분석

Aglets 이동 에이전트 시스템은 에이전트 실행 모듈뿐만 아니라 작업 처리 결과를 모두 포함하며 각 노드를 이주한다. 따라서 에이전트 실행 모듈의 크기

표 1. 시스템의 운영 환경

기종	CPU	Memory	OS	IP
Pentium IV	2.0GHz	512 Mbytes	Windows 2000 Server	142
Pentium IV	1.2GHz	512 Mbytes	Windows XP	144
Pentium III	850MHz	512 Mbytes	Windows 2000 Professional	143
Pentium III	800MHz	384 Mbytes	Windows 2000 Professional	148
Pentium II	750MHz	256 Mbytes	Windows 2000 Professional	146
Pentium II	700MHz	256 Mbytes	Windows 2000 Professional	153,152

가 커져 이주 시간이 많이 걸릴 수 있고, 또한 작업 처리 결과양이 많아 에이전트 크기 증가로 이주 시간이 증가할 수 있다. 이로 인해 그림 13에 보여진 에이전트 이주 시간 측정 결과와 같이 지속적으로 이주 시간이 커진다는 것을 알 수 있다. 각 이주 시간이 비례적으로 증가하지 않는 이유는 에이전트 결과양의 차이에 의해 이주 시간의 증가량이 차이가 난다. 그러나 최적 경로 탐색 이동 에이전트 모델은 에이전트 실행 모듈이 분리되고, 작업을 처리한 결과를 가지고 이주를 수행하지 않음으로 항상 일정하게 에이전트 크기가 유지된다. 따라서 그림 13과 같이 측정된 에이전트의 이주 시간은 최대 50m/s~최소 6m/s까지의 시간 차이로 유지된다. 실제, 에이전트 크기가 일정함으로 이러한 차이가 발생해야 하지 않으나, 각 시스템의 사양에 따라 구현된 에이전트를 실행하는데 걸리는 시간과 각 시스템의 시스템 시간 차이에 의해 이러한 시간차가 나타난다.

4.2 네트워크 트래픽에 따른 이주 시간 분석

네트워크 트래픽에 따른 이주 시간 분석은 Aglets 이동 에이전트 시스템과 제안된 이동 에이전트 모델에서 라우팅 테이블의 각 노드로 이동 에이전트를 이주시킬 때, 각 노드 경로 상에 발생하는 네트워크 트래픽에 따른 에이전트 이주 시간의 변화를 측정하여 두 시스템의 성능을 평가한다. 또한, 이동 에이전트의 이주 시간 변화를 분석하여 제안된 에이전트 이주 기법의 효율성을 평가한다.

네트워크 트래픽에 따른 이주 시간을 측정하기 위해 이동 에이전트는 이전 실험과 같이 7개의 각 노드로 이주를 수행한다. 각 노드 경로 상에는 불규칙적인 네트워크 트래픽이 발생하고 있는 상태로 이동 에이전트가 각 노드를 이주할 경우, Aglets 이동 에이전트 시스템은 고정된 라우팅 스케줄에 따라 이주를 수행하는 반면 제안된 이동 에이전트 모델은 트래픽 감지를 통해 최소의 네트워크 소요시간을 갖는 경로를 탐색하여 이주를 수행한다. 실험조건은 각 노드 경로 상에는 불규칙적인 네트워크 트래픽이 발생하는 상태이다. 단, Aglets 이동 에이전트 시스템과 제안된 이동 에이전트 모델의 이동 에이전트가 동일 노드를 이주할 때 노드 경로 상의 트래픽 크기는 같다.

Aglets 이동 에이전트 시스템에서의 이주 시간은 앞서 측정한 에이전트 크기에 따른 이주 시간 보다 더 걸리게 된다. 이는 실제 에이전트의 크기에 따라 더 많은 트래픽의 영향을 받기 때문에 기존에 측정한 이주 시간보다 더 크게 측정된다. 그러나 최적 경로 탐색 이동 에이전트 모델에서의 트래픽에 따른 이주 시간은 트래픽 탐색 시간의 소요로 Aglets 에이전트 시스템의 이주 시간 보다 효율적이라고 할 수 없다. 실제 실험 환경이 소규모의 네트워크 환경임을 감안한다면 원격지의 노드를 이주 시에는 보다 큰 시간 차이를 가질 수 있다. 그림 14에서 측정된 이주 시간은 점차 시간이 줄어드는 현상을 나타나는데, 이는 에이전트가 계속 이주를 수행하면서 라우팅 테이블의 객체 참조자 리스트가 삭제됨으로 노드 탐색 시간이 줄어들기 때문이다.

위의 두 가지 경우의 실험과 같이 Aglets 이동 에

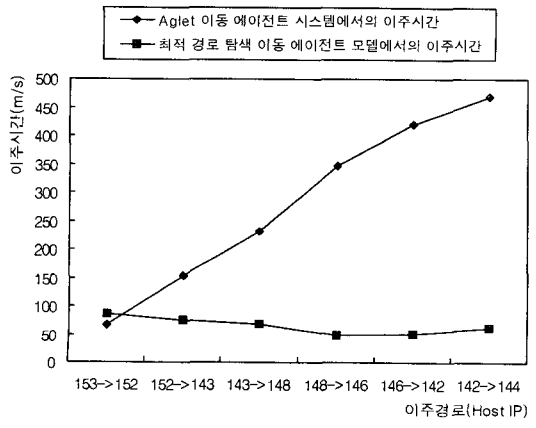


그림 13. 에이전트 크기에 따른 이주 시간 변화

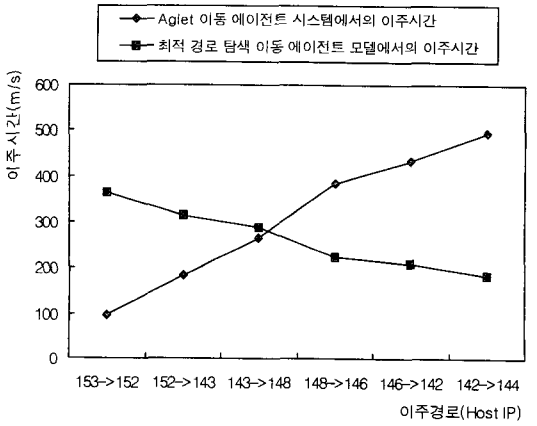


그림 14. 네트워크 트래픽 크기에 따른 이주 시간 변화

이전트 시스템은 이동 에이전트가 각 노드를 이주하며 작업을 처리한 결과를 포함하고 이주를 수행함으로 일정한 에이전트의 크기를 유지하는 제안된 에이전트 모델의 이동 에이전트보다 이주시간이 많이 걸리며, 또한, 네트워크 트래픽에 따른 경우, Aglets 이동 에이전트 시스템은 고정적인 라우팅 스케줄에 따라 각 노드를 이주하며 작업을 처리함으로 네트워크 트래픽의 영향을 많이 받을 수 있는 반면, 제안된 에이전트 모델은 동적인 라우팅 스케줄에 따라 각 노드를 이주함으로 네트워크 트래픽의 영향을 보다 적게 받는다. 하지만, 제안된 에이전트 모델은 최적 이주 경로를 탐색하는 시간이 걸리기 때문에 소규모의 네트워크 환경에서 이용하는 것보다 거리가 먼 원격지의 노드 간을 이주하는데 효율적이다. 본 실험은 제한된 환경에서 수행되었음으로 원격지의 노드 간 이주 시간을 측정한 결과 보다 큰 차이를 측정할 수 없었다.

5. 결론 및 향후 연구

본 논문에서는 네트워크 트래픽 감지를 이용한 이주 노드들에 대한 최적 경로 탐색을 통해 사용자 요구 처리에 있어 최소의 순회 이주 시간을 가지는 CORBA 기반의 이동 에이전트 모델을 설계 및 구현하였다. 제안된 에이전트 모델에서는 자율적인 작업 처리 능력을 가진 이동 에이전트가 분산된 노드들을 이주하며 작업을 처리할 때, 최소의 네트워크 지연 시간을 갖도록 라우팅 스케줄을 계획하여 에이전트를 이주시키기 위한 이동 에이전트의 이주 기법들을 제안하였으며, 제안된 이주 기법들에서 필요로 하는 네트워크 트래픽 측정 및 네트워크 장애 검출을 위해, Ping 프로그램을 이용하여 패킷의 평균 왕복 시간과 패킷 분실 수에 대한 정보들을 획득함으로써 트래픽 크기 측정 및 통신망 결손에 대응하는 방법을 제시하였다. 따라서 이러한 이주 기법들이 제공하는 이동 에이전트 모델은 분산 환경에서 많은 양의 정보들을 보다 효과적으로 빠르게 검색하여 사용자들에게 안정되고 정확한 정보 서비스를 제공함으로써 정보 서비스의 질을 향상시킬 수 있다.

향후 연구과제로는 제안된 이동 에이전트 모델을 기반으로 하는 응용 시스템의 개발이 요구되며, 각 이동 에이전트 플랫폼에서의 에이전트들에 대한 보안 관리를 위해 시스템 모니터링 에이전트를 확장한 보안 에이전트의 개발이 필요하다. 또한, MAFFinder의 네임 스페이스내 메타 데이터 테이블의 관리에 있어 캐시 기법을 적용하여 보다 효율적으로 메타 데이터를 관리하기 위한 연구가 지속적으로 수행되어야 한다.

참 고 문 헌

[1] J. Baumann, "A Protocol for Orphan Detction and Termination in Mobile Agent Systems", TR-1997-09, Stuttgart Univ. 1997.

[2] A. Yariv and D. B. Lange, "Agent Design Patterns : Elements of Agent Application Design", Second International Conference on Autonomous Agents(Agents 98), 1998.

[3] OMG, "Mobile Agent System Interoperability Facilities Specification", OMG TC Document orbos /97-10-05, 1997.

[4] IKV++, "A CORBA environment supporting Mobile Agent", IKV++ GmbH, 1999.

[5] S. Choy and T.Magedanz, "Grasshopper Technical Overview", IKV++ GmbH, 1999.

[6] General Magic, "Odyssey, http://www.genmagic.com/agents/

[7] ObjectSpace, "ObjectSpace Voyager, GeneralMagic Odyssey, IBM Aglets : A Comparison", VoyagerTM, 1997.

[8] K. Rothermel and M. Strasser, "A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents", Proceeding of the 17th IEEE SRDS98, 1998.

[9] K. A. Baharat and L. Cardelli, "Migratory Applications", Proceedings of the 8th annual ACM symposium on User interface and software technology, November 1995.

[10] J. Vitek and Christian Tschudin, "Mobile Object Systems : Towards the Programmable Internet", Springer-Verlag, April 1997.

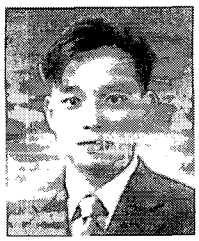
[11] Lars Hagen, Markus Breugst, and Thomas Margendanz, "Impact of Mobile Agent Technology on Mobile Communication System Evolution", IEEE Personal Communications, pp. 56-68, 1998.

[12] Ahmed Karmouch, "Mobile Software Agents for Telecommunications", IEEE Communication Magazie, pp. 24-25, 1998.
[13] 전병국, 최영근, "이동 에이전트를 위한 효율적인 이주 정책 설계 및 구현", 한국정보처리학회 논문지, 제6권, 제7호, pp. 1770-1776, 1999.
[14] Robert S.G, "Agent Tcl : A flexible and secure mobile-agent system", TR98-327, Dartmouth Col. June 1997.
[15] OMG, "Agent Technology Green Paper", Agent Platform Special Interest Group, http://www.objs.com/agent/index.html 2000.
[16] Paolo Bellavista, Antonio Corradi, and Cesare Stefanelli, "A Mobile Agent Infrastructure for the Mobility Support", Proceedings of the 2000 ACM symposium, ACM Press, USA, pp. 539-545, 2000.
[17] David Chess, Benjamin Grosf, Colin Harrison, and David Levine, Colin Parris, and Gene Tsudik, "Itinerant Agents for Mobile Computing", IEEE Personal Communication Magazine, Vol.2, No.4, pp. 34-59, 1999.
[18] Vn Anh Pham and Ahmed Karmouch, "Mobile software Agents: An Overview", IEEE Communication Magazine, pp. 26-37, 1998.
[19] D.B.Lange and M.Oshima, "Programming and deploying Java Mobile Agents with Aglets", Addison Wesley Press, 1998.
[20] IBM, "The Aglets Workbench", http://www.trl.ibm.co.jp/aglets/

김 광 종

1993년 군산대학교 컴퓨터정보과학과 이학사
1999년 군산대학교 컴퓨터정보과학과 이학석사
2004년 군산대학교 컴퓨터정보과학과 이학박사

관심분야 : 에이전트, 분산객체시스템, 능동 데이터베이스



고 현

2001년 군산대학교 컴퓨터정보과학과 이학사
2003년 군산대학교 컴퓨터정보과학과 이학석사
2004년 군산대학교 컴퓨터정보과학과 박사과정

관심분야 : 에이전트, 분산객체시스템, 멀티미디어 데이터베이스

김 영 자

1993년 군산대학교 컴퓨터정보과학과 이학사
2000년 순천대학교 컴퓨터정보과학과 이학석사
2004년 군산대학교 컴퓨터정보과학과 박사수료

관심분야 : 에이전트, 분산객체시스템, CDN

이 연 식

1982년 전남대학교 전자계산학과 이학사
1984년 전남대학교 전자계산학과 이학석사
1994년 전북대학교 전산응용공학전공 공학박사
1995년~1997년 군산대학교 교무부처장
1997년~1998년 University of Missouri 교환교수
1999년~2001년 군산대학교 전자계산소 소장
1986년~현재 군산대학교 컴퓨터정보과학과 교수
관심분야 : 번역기 이론, 객체지향시스템, 능동시스템, 지능형 에이전트