대규모 이동 에이전트 시스템을 위한 적응적 에이전트간 통신 프로토콜

안 진 호†

요 약

본 논문에서는 이동 에이전트의 홈 노드에 대한 의존성을 피하면서 각 서비스 노드에 의해 유지되는 에이전트 위치정보량과 메시지 전달 시간을 매우 줄이는 적응적 에이전트간 통신 프로토콜을 제안한다. 이러한 목적을 달성하기 위해, 제안된 프로토콜은 각 이동 에이전트가 자율적으로 단지 자신이 방문한 노드들 중 일부분에게만 그 에이전트의 위치정보를 남겨두도록 한다. 또한, 이 프로토콜은 각 서비스 노드의 스마트 에이전트 위치 캐쉬에 각 에이전트의 위치관리자 식별자를 유지하게 함으로써, 노드의 캐쉬 갱신 횟수를 매우 줄일 수 있다. 본 논문에서 수행한 시뮬레이션에서는 제안된 프로토콜이 기존 프로토콜에 비해 메시지 전달 비용을 76% ~ 80% 정도 줄이고, 각 서비스 노드가 유지해야 할 에이전트 위치정보량을 76% ~ 79% 정도 줄인다는 것을 보여준다.

키워드 : 이동 에이전트 시스템, 확장성, 신뢰성, 디렉토리 서비스, 메시지 전달 서비스

Adaptive Inter-Agent Communication Protocol for Large-Scale Mobile Agent Systems

Jinho Ahn†

ABSTRACT

This paper proposes an adaptive inter-agent communication protocol to considerably reduce the amount of agent location information maintained by each service node and the message delivery time while avoiding the dependency of the home node of a mobile agent. To satisfy this goal, the protocol enables each mobile agent to autonomously leave its location information only on some few of its visiting nodes. Also, it may significantly reduce the agent cache updating frequency of each service node by keeping the identifier of the location manager of each agent in the smart agent location cache of the node. Simulation results show that the proposed protocol reduces about 76% ~ 80% of message delivery overhead and about 76% ~ 79% of the amount of agent location information each service node should maintain compared with the traditional one.

Key Words : Mobile Agent System, Scalability, Reliability, Directory Service, Message Delivery Service

1. 서 론

PDA와 셀룰러폰과 같은 무선 디바이스들과 그리드 및 유비쿼터스 컴퓨팅과 엑티브네트워크와 같은 새로운 인터넷 기반 기술들이 급속도로 출현함에 따라, 현재의 컴퓨팅 환경은 매우 복잡해지고 있다[2, 3, 9, 10]. 또한, 사용자와 디바이스들의 이동성으로 인해 그들의 소프트웨어가 다양한 능력과 로컬 자원들을 제공하는 동적인 컴퓨팅 환경에서 수행되어야 한다. 소프트웨어 개발 측면에서 설계할 애플리케이션 프로그램이 수행될 동적 런타임 환경에 관한 완전히 정확한 정보를 미리 얻을 수 없기 때문에, 앞에서 언급한 이러한 이슈들은 애플리케이션 개발을 매우 어렵게 한다. 이를 위해 소프트웨어 컴포넌트들이 런타임시 그들의 수행 환경에 적응할 수 있도록 하는 새로운 미들웨어 플랫폼의 개발이 필수적이다.

이동 에이전트 기술은 앞에서 언급한 컴퓨팅 환경의 복잡성과 다양성을 고려하기 위한 잠재적인 매개체로써 매우 높은 관심을 끌고 있다[2, 3, 9-11]. 이동 에이전트는 한 네트워크에서 노드와 노드간을 이동하면서, 그 노드에서 제공하는 서비스를 이용하여 사용자 대신 특정된 일을 수행하는 능동적이고 자율적인 프로그램이다. 그러나, 이동 에이전트

※ 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음. (KRF-2004-003-D00281)
† 종신회원 : 경기대학교 정보과학부 전자계산학과 조교수
논문접수 : 2006년 2월 23일, 심사완료 : 2006년 6월 22일

시스템의 규모가 급속도로 커짐에 따라 확장성이 가장 중요한 이슈가 되었다. 이러한 확장성을 제공하기 위해, 이동 에이전트 시스템에서 구성요소들 중 에이전트 통신 및 배치, 모니터링 및 보안 등에 대한 재설계가 요구된다. 특히, 이 구성요소들 중 신뢰성 있는 에이전트간 통신메커니즘을 확장적으로 설계하는 것이 이 시스템에서 매우 필수적이다.

일반적으로 에이전트간 통신 메커니즘의 구성요소들로는 크게 두 가지로 나뉘어 진다. 첫 번째 요소는 이동 에이전트의 위치에 관한 정보를 유지하기 위한 디렉토리 서비스(directory service)이고, 두 번째 요소는 디렉토리 서비스를 사용하여 메시지를 해당 이동 에이전트에게 신속하고 올바르게 전달해주는 메시지 전달 서비스(message delivery service)이다[1, 12]. 그러나, 이동 에이전트 시스템에서는 기존의 고정 에이전트(stationary agent)간 통신 메커니즘과는 달리 에이전트 이동성(agent mobility)이라는 특성을 가짐으로써 다음과 같이 추가적인 두 가지 요구사항들을 만족해야 한다.

■위치 독립성(location independence)

한 이동 에이전트가 자신의 작업을 수행하기 위해, 자율적으로 호스트들 간에 이동한다. 이러한 특성 때문에 임의의 에이전트가 통신하고자 하는 다른 에이전트의 현재 위치를 메시지 송신 전에 미리 알아야 한다는 것은 매우 실용적이지 못하다. 따라서, 상대방 에이전트의 현재 위치를 미리 알지 못한다 할지라도 에이전트간 통신을 원활히 수행할 수 있도록 해야 한다.

■신뢰성 있는 에이전트간 통신(reliable inter-agent communication)

이동 에이전트 시스템에서 메시지 전달과정과 에이전트 이주과정은 서로 비동기적으로 수행된다. 이러한 특성 때문에 임의의 에이전트 이동시 그 에이전트로의 메시지 손실(message loss)이 발생할 수 있다. 그러므로, 목적 에이전트(target agent)가 자주 호스트간 이주하더라도 그 에이전트로 송신되는 메시지들은 정확하게 전달될 수 있도록 해야 한다.

이러한 에이전트 이동성을 고려하는 제안된 대표적인 기존 이동 에이전트간 통신기법들은 다음과 같다. 첫 번째로, 브로드캐스트 기반(broadcast-based) 기법[14]은 투명하고 신뢰성 있는 에이전트간 통신을 보장하고, 다수의 에이전트들을 하나의 그룹으로써 멀티캐스트 통신을 가능하게 해준다. 그러나, 임의의 메시지에 대한 목적 에이전트를 위치시키기 위해, 이 기법은 네트워크에서 그 에이전트가 방문했던 모든 노드들과 통신해야 한다. 따라서, 브로드캐스트 기반 기법은 이러한 특성에 의해 대규모 분산 에이전트 기반 시스템에서 매우 많은 트래픽을 발생시킬 수 있으므로 실용적이지 못하다. 두 번째로, 홈 에이전트 기반(home agent-based) 기법[11]에서는 각 이동 에이전트의 위치정보가 자신이 생성된 호스트에 존재하는 홈 에이전트에 의해 관리된다. 즉, 이동 에이전트가 홈 호스트가 아닌 다른 외부 호스트로 이동할 때마다, 해당 홈 에이전트에게 자신의 위치를 알려주어야 한다. 만약, 송신자 에이전트가 그 이동 에이전트로 메시지를 보낸다면, 그 메시지는 먼저 해당 홈 호스트로 전달된다. 이 경우, 홈 에이전트는 그 메시지를 해당 이동 에이전트로 전달한다. 그러나, 이 기법에서 각 이동 에이전트로 향하는 모든 메시지들이 반드시 해당 홈 호스트를 거쳐서 전달된다. 따라서, 이동 에이전트 시스템의 규모가 커지는 경우, 이 기법은 홈 호스트에 높은 부하를 발생시킬 수 있다는 단점을 가지게 된다. 또한, 홈 호스트가 네트워크로부터 단절되는 경우 자신이 관리하는 모든 이동 에이전트들의 통신이 불가능해진다. 마지막으로, 포워딩 포인터 기반(forwarding pointer- based) 기법[8, 12, 15, 16]에서는 각 이동 에이전트의 위치정보가 특정한 홈 에이전트에 의해서만 관리되는 것이 아니라, 그 에이전트가 다른 호스트로 이동할 때마다 이전에 방문한 호스트에 자신의 위치정보(포워딩 포인터)를 남긴다. 따라서, 어떤 송신자 에이전트가 임의의 이동 에이전트로 메시지를 보낼 경우, 이 이동 에이전트가 현재까지 방문했던 호스트들에 유지되고 있는 포워딩 포인터를 통해서 그 메시지가 전달된다. 그러므로, 이 기법은 홈 에이전트 기반 기법에서의 홈 호스트의 확장성 문제와 네트워크 단절 문제를 해결할 수 있다. 그러나, 이 기법은 메시지 전송 시 해당 이동 에이전트가 방문했던 모든 호스트를 통해야만 메시지가 전달되므로, 자주 이주하는 이동 에이전트(highly mobile agent)에 대한 메시지 포워딩 비용이 매우 높아질 수 있다. 또한, 이동 에이전트 수가 급속히 증가하는 경우 각 호스트가 관리해야 하는 포워딩 포인터 수도 그만큼 늘어난다. 이는 이동 에이전트의 위치정보 저장소의 부족 및 관리비용의 증가를 발생시킬 수 있다.

그러나, 계속적으로 이동 에이전트 시스템이 대중화됨에 따라 시스템의 네트워크 규모가 매우 커지고, 사용되는 이동 에이전트의 수가 급속도로 증가함에 따라, 앞에서 언급한 단점들을 가진 각 기존 방법에 비해 보다 확장적인 이동 에이전트간 통신 메커니즘이 설계될 필요가 있다. 따라서, 본 논문에서는 이러한 기존 기법들의 단점을 해결하기 위해 이동 에이전트 시스템의 환경적 변화에 효율적으로 적응할 수 있는 확장적이고 신뢰성 있는 에이전트간 통신 프로토콜을 제안한다. 이 프로토콜은 각 이동 에이전트가 자율적으로 자신의 위치정보를 방문노드들 중 일부 노드들에게만 유지하게 함으로써, 기존 프로토콜들에 비해 메시지 전달 시간과 위치정보유지 비용을 매우 줄일 수 있다. 또한, 제안하는 프로토콜은 홈 에이전트 기반 기법과 달리 에이전트 위치갱신 및 메시지 전달 시 홈 노드의 중앙 집중적인 수행형태를 피할 수 있다.

본 논문의 이후 구성은 다음과 같다. 먼저, 2절에서는 본 논문에서의 이동 에이전트 시스템 모델을 기술하고, 3절에서는 관련연구 기술에 대해 설명한다. 4절에서는 확장적이고 신뢰성 있는 에이전트간 통신 프로토콜을 제안한다. 5절

과 6절에서는 각각 제안된 프로토콜의 성능평가 결과를 설명하고, 본 논문의 결론을 제시한다.

2. 시스템 모델

본 논문에서 가정하는 이동 에이전트 시스템에서는 어떠한 전역 메모리와 전역 클럭(global clock)을 가지고 있지 않다. 또한, 이 시스템은 비동기적이다: 모든 에이전트는 각자의 속도로 수행하고, 에이전트간 메시지 전송시간은 유한하지만 임의적이다. 시스템은 n개의 에이전트 서비스 노드들로 구성된다. 각 서비스 노드는 에이전트들이 안전하게 수행할 수 있는 환경을 제공하고, 그 서비스 노드의 로컬 자원들을 방문 에이전트들이 제한적이지만 접근할 수 있도록 한다. 한 에이전트는 최초에 홈 노드(home node)라 하는 서비스 노드에서 생성되어 그 노드로부터 유일한 식별자를 얻는다. 따라서, 각 에이전트는 자신의 로컬 식별자와 자신의 홈 노드 식별자를 결합하여 사용함으로써, 시스템 내에 전역적으로 유일한 객체로써 식별된다. 한 에이전트가 시스템 내에서 이주할 때, 자신의 코드 및 상태정보가 캡쳐되어 다음 노드로 전달된다. 그 노드에 도착한 후, 해당 이동 에이전트는 재생성되어 다른 에이전트들과 상호작용하면서 자신의 일을 수행한다. 사용자를 대신하여 할당된 작업을 수행하기 위해, 한 이동 에이전트 a는 자신의 여정에 따라 k(k>0)개의 서비스 노드들 I_a=[N_{home}, N_1,..., $N_{(k-1)}$]에서 인덱스 순서대로 이주하면서 수행한다. 여기서 에이전트의 여정은 해당 에이전트가 자신이 생성된 노드에서 작업을 시작하기 전에 정적으로 결정될 수도 있고, 이주 혹은 수행 중 동적으로 결정될 수 있다. 통신채널은 비동기적 메시지 전달을 가능하게 하고, 신뢰성 있고, 선입선출 형이다. 또한, 네트워크 파티션은 발생하지 않는다고 가정한다. 이러한 채널을 통해 에이전트들은 이주할 수 있으며, 메시지들이 전달될 수 있다. 그러나, 본 논문에서는 각 서비스 노드가 고장나지 않는다고 가정하고, 서비스 노드와 이동 에이전트의 고장에 대한 고려는 본 시스템 모델의 범위를 벗어나며, 이러한 부분은 다른 연구[13, 17]에서 다루어지고 있다.

3. 관련 연구

Mole[4]에서는 임의의 에이전트가 결코 다른 에이전트와 통신 중에 이주하지 않는다고 가정한다. 따라서, 통신하는 에이전트들이 이주하는 동안, 그들간의 통신은 묵시적으로 종료되어진다. 이 시스템에서는 보다 빠른 통신을 위해 각 에이전트의 포인터들을 유지한다. 그러나, 이것은 단지 고아 감지(orphan detection) 프로토콜의 컨텍스트에서만 사용된다. 또한, Mole은 이동하지 않는(non-mobile) 에이전트들 간의 정보 교환을 위한 그룹통신을 지원한다.

Concordia[19]에서는 이벤트들을 통한 에이전트간 통신을 제공한다. 각 에이전트가 이벤트 통신을 하려면, 반드시 해당 이벤트 관리자에게 등록하는 과정이 필요하다. 이후에, 송신 에이전트가 이벤트 관리자와 연결하여 자신의 요구를 포함한 이벤트를 포스트 한다. 그 이벤트 관리자가 특정 이벤트를 수신하자마자, 그 관리자는 등록된 모든 에이전트에게 해당 이벤트를 알려준다. 한 에이전트가 다른 노드로 이주하더라도, 그 에이전트는 해당 이벤트 관리자로부터 여전히 이벤트들을 수신할 수 있다. 각 에이전트는 다수의 이벤트 관리자들에게 등록할 수 있다. 그러나, 에이전트간 통신이 가능하기 위해서는, 수신 에이전트가 정확히 이벤트 관리자 위치를 알고 있어야만 한다.

기존연구 [18]에서는 한 메시지의 손실을 감지하고 그 메시지를 재전송하기 위해 TCP와 유사한 슬라이딩 윈도우 프로토콜을 사용하는 송신 기반 메시지 전달 메커니즘을 제안하였다. 몇 번의 재송신 후에 그 송신자는 위치서버와 접촉한 후, 그 메시지를 새로운 목적지에 전달한다. 그러나, 그 메커니즘은 메시지 라우팅과 에이전트 이주를 동기화할 필요가 있다.

Murphy와 Picco는 브로드캐스트 기반 이동 에이전트 통신 프로토콜[14]을 제안하였다. 이 프로토콜은 투명하고 신뢰성있는 에이전트간 통신을 제공하고, 여러 에이전트들로 구성된 에이전트 그룹을 위한 멀티캐스트 통신을 가능하게 한다. 그러나, 메시지의 목적지를 위치시키기 위해, 그 프로토콜은 네트워크 내에 있는 모든 방문 노드들에게 질의해야 한다. 따라서, 이 프로토콜은 높은 트래픽을 발생시킴으로써 대규모 이동 에이전트 시스템에 적용하기에 적합하지 않다.

Belle는 계층적 서버 구조로 구성된 위치 디렉토리를 형성시키는 계층적 구조 기반 메커니즘[6]을 제안하였다. 각 레벨에서 위치 서버는 하위 수준 객체 위치정보를 유지한다. 각 객체의 정보는 하위 수준 위치서버에서의 한 엔트리를 가리키는 포인터이거나 에이전트의 실제 현재 위치이다. 그러나, 이러한 계층적 구조는 동적인 네트워크 환경에서 쉽게 형성될 수 없다. 또한, 이 메커니즘에서 불필요한 홉들이 계층적 구조에 포함될 수 있다.

Cao와 Feng은 위치 독립적이고 신뢰성 있는 메시지 전달을 제공하는 메일박스 기반 메커니즘[7]을 제안하였다. 그 메커니즘은 임의의 메시지가 해당 에이전트에게 전달되기 전에 기껏해야 한 번 포워딩될 수 있도록 한다. 또한, 각 메일박스가 해당 에이전트에게 이주되어야 하는지를 자율적으로 결정함으로써, 임의의 에이전트 이주는 자신의 메일박스 이주와 분리될 수 있다. 그러나, 메일박스로의 메시지 전달에 대한 불확실성은 불필요한 빠른 폴링(early polling)을 발생시킬 수 있다. 반면에, 긴급 메시지들이 정각에 메일박스로 전달될지라도, 목적지 에이전트의 폴링시간에 따라 그 메시지들은 매우 늦게 해당 에이전트에게 전달될 수 있다. 또한, 각 메일박스가 이주할 때마다, 자신의 새로운 위치정보를 그 메일박스가 이전에 방문하였던 모든 노드에게 브로드캐스트해야 한다. 이러한 수행형태는 대부분의 에이전트들이 이동성이 높은 경우, 상당히 높은 네트워크 트래픽을 발생시킬 수 있다.

4. 확장적 신뢰성 있는 에이전트간 통신 프로토콜

본 논문에서 설계하는 에이전트간 통신 프로토콜은 (그림 1)과 같은 두 가지 특성을 가지고 있다. 이러한 두 가지 특성을 만족하기 위해서는 설계될 프로토콜이 다음과 같은 두 가지 형태로 수행되어야 한다.

- 위치 갱신 및 메시지 전달 비용 등과 같은 이동 에이전트의 우선정책에 따라 자율적으로 자신이 방문하는 모든 노드들 중 일부노드들만 해당 에이전트의 포워딩 포인터를 유지하도록 함.

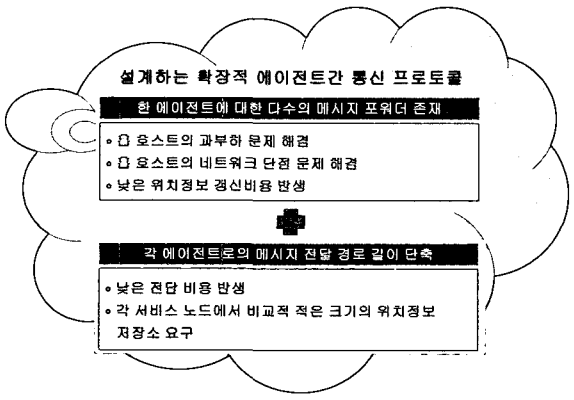
- 한 이동 에이전트의 포워딩 포인터를 유지하는 서비스 노드들(포워더(forwarder)) 중 가장 최근에 방문한 노드(위치관리자(location manager))만이 그 에이전트가 현재 수행하고 있는 서비스 노드의 위치정보를 관리함.

이러한 프로토콜의 수행형태에 대한 이해를 돕기 위해, (그림 2)에서는 한 에이전트 a가 홈 노드 N_{home}으로부터 노드번호 순서대로 12개의 노드들을 방문하는 예를 보여준다. 이 예를 통해 기존의 홈 기반 기법, 포워딩 포인터 기반 기법과 본 논문에서 제안하는 기법을 비교하고자 한다. 먼저, (그림 2)(a)와 같이 홈 기반 기법에서는 단지 홈 노드 N_{home}만이 에이전트 a의 포워딩 포인터를 가지는 포워더이며 또한 위치 관리자가 된다. 따라서, 이 기법에서는 에이전트 a가 이주할 때마다 자신의 위치정보를 노드 N_{home}에만 항상 등록한다. 또한, 다른 에이전트들이 에이전트 a에게 메시지를 송신하는 경우, 반드시 노드 N_{home}를 통해서만 a에게 전달될 수 있다. 따라서, 이 기법은 (그림 1)에서의 첫 번째 특성을 만족하지 못한다.

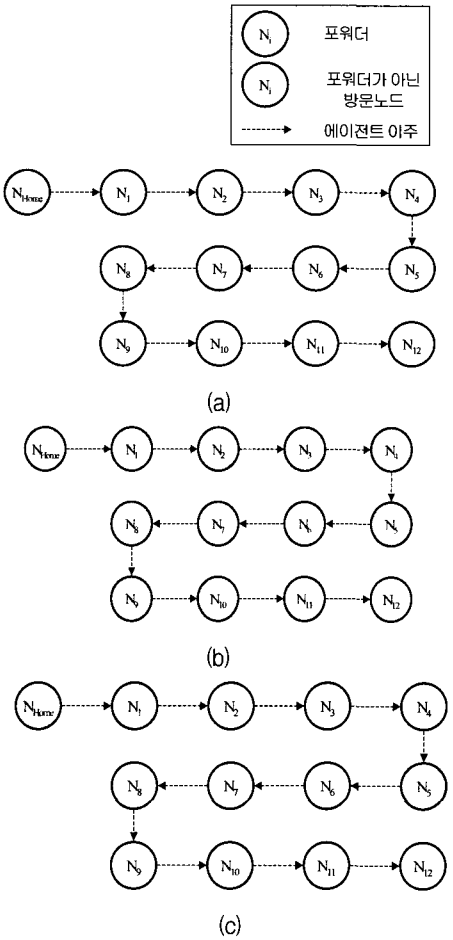
두 번째로, (그림 2) (b)와 같이 포워딩 포인터 기반 기법에서는 에이전트 a가 현재 수행하고 있는 마지막 노드 N_{12}를 제외한 홈 노드를 포함한 모든 방문노드들이 a의 포워딩 포인터를 유지한다. 즉, 에이전트 a가 새로운 노드로 이주하면, 그 이전 방문노드에게 자신의 새 위치정보를 등록한다. 따라서, 다른 에이전트들이 에이전트 a에게 메시지를 송신하는 경우, 홈 노드로부터 11개의 방문노드들을 거쳐서 노드 N_{12}에 전달된다. 따라서, 이 기법은 (그림 1)에서의 두 번째 특성을 만족하지 못한다.

마지막으로, (그림 2) (c)와 같이 본 논문에서 설계하는 확장적 디렉토리 서비스 프로토콜에서는 에이전트 a가 서비스 노드를 방문하면서 자신의 우선정책에 의해 효율성을 극대화할 수 있도록 자율적으로 적은 수의 포워더들 N_1, N_5, N_{10}을 선택한다. 또한, 현재 에이전트 a가 노드 N_{11}에서 노드 N_{12}로 이주하는 경우, a는 자신의 현재 위치정보를 최근에 방문한 포워더인 노드 N_{10}에게 등록한다. 따라서, 이 기법은 (그림 1)에서의 두 가지 특성들을 모두 다 만족할 수 있다.

제안하는 프로토콜에서 이동 에이전트가 홈 노드에게 자신의 위치정보를 전혀 갱신하지 않는다면, 그 에이전트로 다른 에이전트들이 최초에 메시지를 송신하는 경우, 메시지 포워딩 경로가 길어짐으로 메시지 전달 비용이 증가한다. 그러나, 이 비용을 줄이기 위해 홈 노드로 자주 갱신한다면, 갱신비용이 높아진다. 따라서, 홈 노드로의 위치정보 갱신간격은 해당 이동 에이전트가 자신의 이동주기가 높아지면 짧게하고, 낮아지면 길게함으로써 이동성 정도에 따라 효율적으로 적응할 수 있도록 설정한다.



(그림 1) 설계하는 에이전트간 통신 프로토콜의 두 가지 특성



(그림 2) 에이전트 a가 홈 노드 Nhome에서부터 N1을 거쳐 N12까지 노드번호 순서대로 이주하는 예

4.1 디렉토리 서비스

앞에서 언급한 특성들을 가지는 디렉토리 서비스 프로토콜은 각 서비스 노드마다 <표 1>과 <표 2>에서 소개되는 데이터 구조를 유지해야 한다.

제안하는 디렉토리 서비스 프로토콜이 어떻게 이러한 특성들을 만족할 수 있는지를 (그림 3)를 통해 설명하고자 한다. 이 그림은 에이전트 a가 자신의 여정에 따라 자신의 홈 노드로부터 노드 N_1에서 노드 N_5까지 방문하는 동안, 에이전트 a의 위치갱신 시 발생하는 노드간의 메시지 상호작용과 각 노드에 의해 유지되는 위치 정보를 보여준다. (그림 3) (a)에서 첫 번째 단계로써 에이전트 a가 N_{home}에서 생성되어 그 에이전트를 위한 원소인 (id_a, Home, 0)을 자신의 $RunningAgents_{home}$에 저장한다. 두 번째 단계에서 에이전트 a가 부분적인 작업을 수행한 후, 노드 N_1으로 이주하고자 한다면, a는 N_{home}이 현재 자신의 위치관리자이고 노드 N_1으로 이주 중이라는 것을 나타내는 a를 위한 원소인 (id_a, 1, 1, true, true)을 $AgentLocs_{home}$에 저장한다. 이후에 N_{home}은 자신의 노드 식별자와 에이전트 a의 타임스탬프를 에이전트와 함께 노드 N_1으로 전달한다. 이들을 수신한 노드 N_1은 a의 타임스탬프를 1만큼 증가시킨다. 이 때, a가 노드 N_1이 자신의 위치지정자가 되길 원하기 때문에, 세 번째 단계로써 a의 위치정보 (id_a, 1, 1)를 $RunningAgents_1$에 삽입한다. 동시에 N_1은 자신이 이후부터 a의 위치관리자 역할을 수행한다는 것을 N_{home}에게 알려주기 위해 a의 타임스탬프를 포함한 메시지 changeImngr를 N_{home}에게 송신한다. 그 메시지를 수신한 N_{home}은 a의 위치정보를 $AgentLocs_{home}$에 갱신하고, a의 원소에서 두 필드인 ismanaging_f와 ismigrating_f를 모두 false로 갱신한다. 이 경우, 만약 a의 이주과정에 의해 a로의 메시지들이 N_{home}의 메시지 큐에 버퍼링되었다면, 그 메시지들을 노드 N_1으로 전달한다. 네 번째 단계로 a가 자신의 부분적인 작업의 수행을 완료한 후, 노드 N_1으로 이주하고자 한다면, 노드 N_1은 a의 원소인 (id_a, 2, 2, true, true)을 $AgentLocs_1$에 삽입하고, 에이전트 a를 N_2로 전달한다. 이 경우 다섯 번째 단계로 노드 N_2는 a의 타임스탬프를 1만큼 증가시키고, a가 노드 N_2를 단지 방문노드로 원하기 때문에 a의 원소 (id_a, 1, 2)를 $RunningAgents_2$에 삽입한다. 또한, N_2는 a의 타임스탬프를 포함한 메시지 update를 N_1에게 송신함으로써 a의 현재 위치를 위치지정자 N_1에게 등록시킨다. a로의 메시지들이 N_1의 메시지 큐에 버퍼링되었다면, 그 메시지들을 노드 N_2로 전달한다.

(그림 3) (b)는 에이전트 a가 노드 N_2에서 N_3로 이주하는 과정을 보여준다. 이 예에서, N_2는 먼저 N_1에게 지금부터 a의 이주과정이 시작된다는 것을 알리는 메시지 m_initiated를 송신한다. 두 번째 단계로써 그 메시지를 수신한 노드 N_1은 a의 원소에서 필드 중 ismigrating_f를 true로 설정한 후, 메시지 m_reply를 노드 N_2에게 송신한다. 만약, 에이전트 이주과정이 이러한 무효화 과정의 수행없이 시작된다면 어떠한 문제점을 발생시킬 수 있는지를 예를 들어보면 다음과 같다. 이 경우, 노드 N_1이 a로 전달되어야 할 임의의 메시지를 수신한다면, N_1은 현재 a의 이주과정이 현재 수행중인지를 알지 못하기 때문에, 그 메시지를 N_2에게 전달하다. 그러나, a는 현재 노드 N_2에서 수행하지도 않으며, N_2가 a의 포워더가 아님으로 $AgentLocs_2$에 a의 위치정보를 유지하고 있지 않다. 이 경우, 그 메시지는 에이전트 a에게 전달될 수 없다. 그러므로, N_2는 a의 위치관리자로부터 메시지 m_reply를 수신한 후, a를 N_3으로 전달하고, $RunningAgents_2$로부터 a의 원소를 삭제해야 한다. 세 번째 단계로 a의 방문노드 N_3은 a의 타임스탬프를 1 증가시키고, a의 원소 (id_a, 1, 3)을 $RunningAgents_3$에 저장한 후 메시지 update를 N_1에게 송신한다. 그 메시지를 송신한 N_1은 $AgentLocs_1$내 a의 원소를 (id_a, 3, 3, true, false)로 갱신한다.

(그림 3) (c)는 에이전트 a가 노드 N_4에서 N_5로 이주할 때, N_5가 a의 위치관리자로 되는 예를 보여준다. 이 경우, N_5는 a의 위치정보 (id_a, 5, 5)를 생성하고 $RunningAgents_5$에 삽입한 후, N_5가 자신이 이후부터 a의 위치관리자 역할을 수행한다는 것을 이전 위치관리자 N_1에게 알려주기 위한 메시지 changeImngr를 N_1에게 송신한다. 또한, 다른 에이전트가 최초에 N_{home}을 통해 메시지를 a에게 송신하는 경우 발생되는 메시지 전달 시간을 줄이기 위해, 에이전트 a는 자신의 현재 위치를 N_{home}에게 등록할 수 있다. 이러한 목적을 위해, N_5는 메시지 update를 N_{home}에게 송신한다. 만약 a가 N_{home}으로의 이러한 갱신과정이 성능향상에 별로 도움이 되지 않는다고 판단된다면, 그러한 갱신과정을 수행하지 않는다.

에이전트 디렉토리 서비스 프로토콜에서 각 서비스 노드를 위한 프로시저들은 (그림 4)에서 정형적으로 표현된다.

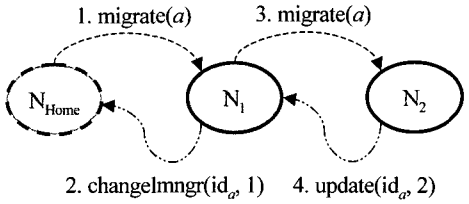
〈표 1〉 RunningAgentsi

정의	노드 N_i에서 현재 수행하는 모든 에이전트의 위치정보를 저장하는 벡터.	
필드 구성	• agent_id:	해당 이동 에이전트 식별자
	• locmngr_n:	에이전트 agent_id가 노드 N_i에서 현재 수행한다고 알고 있는 노드, 즉 위치관리자 식별자
	• agent_t:	에이전트 agent_id의 노드 N_i에서 현재 할당된 타임스탬프

〈표 2〉 AgentLocsi

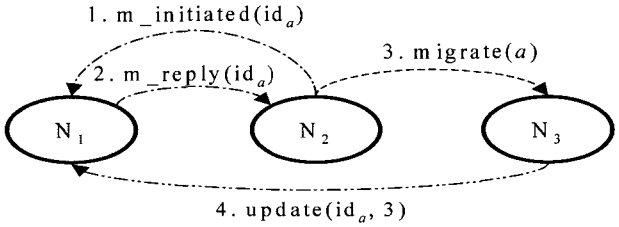
정의	노드 N_i에서 현재 수행하지 않지만, N_i가 포워딩 노드인 모든 이동 에이전트의 위치정보를 저장하는 벡터.	
필드 구성	• agent_id:	해당 이동 에이전트 식별자
	• next_n:	에이전트 agent_id가 현재 수행하거나 그 에이전트의 위치지정자라고 알고 있는 노드의 식별자
	• agent_t:	N_i가 알고 있는 에이전트 agent_id가 수행한 가장 최근노드에서 할당된 타임스탬프
	• ismanaging_f:	N_i이 현재 에이전트 agent_id의 위치지정자인지 (=true) 아닌지(=false)를 나타내는 비트 플래그
	• ismigrating_f:	에이전트가 현재 다른 노드로 이주 중인지 (=true) 아닌지(=false)를 나타내는 비트 플래그

	N_{Home}		N_1		N_2
	$RunningAgents_{Home}$	$AgentLocs_{Home}$	$RunningAgents_1$	$AgentLocs_1$	$RunningAgents_2$
Step 1)	(id_a, home, 0)				
Step 2)		(id_a, 1, 1, true, true)			
Step 3)		(id_a, 1, 1, false, false)	(id_a, 1, 1)		
Step 4)		(id_a, 1, 1, false, false)		(id_a, 2, 2, true, true)	
Step 5)		(id_a, 1, 1, false, false)		(id_a, 2, 2, true, false)	(id_a, 1, 2)



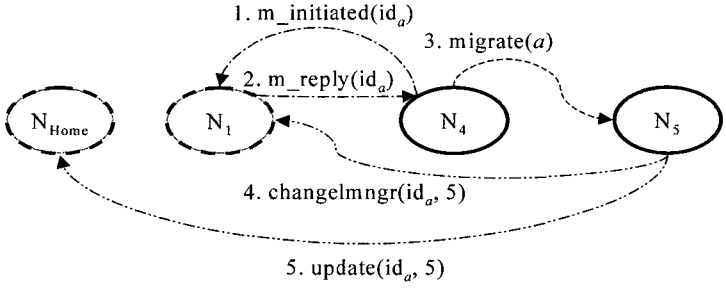
(a) 에이전트 a가 홈 노드로부터 노드 N1과 N2로 이주하는 예

	N_1	N_2	N_3
	$AgentLocs_1$	$RunningAgents_2$	$RunningAgents_3$
Step 1)	(id_a, 2, 2, true, false)	(id_a, 1, 2)	
Step 2)	(id_a, 2, 2, true, true)	(id_a, 1, 2)	
Step 3)	(id_a, 3, 3, true, false)		(id_a, 1, 3)



(b) 에이전트 a가 노드 N_2로부터 N_3로 이주하는 예

	N_{Home}	N_1	N_4	N_5
	$AgentLocs_{Home}$	$AgentLocs_1$	$RunningAgents_4$	$RunningAgents_5$
Step 1)	(id_a, 1, 1, false, false)	(id_a, 4, 4, true, false)	(id_a, 1, 4)	
Step 2)	(id_a, 1, 1, false, false)	(id_a, 4, 4, true, true)	(id_a, 1, 4)	
Step 3)	(id_a, 5, 5, false, false)	(id_a, 5, 5, false, false)		(id_a, 5, 5)



(c) 에이전트 a가 노드 N_4로부터 N_5로 이주하는 예

(그림 3) 에이전트 a가 자신의 여정에 따라 이주 시 위치 갱신하는 예

```
Actions taken when agent a on Ni attempts to migrate to node dest_n
        e ← look up agent a's element from RunningAgentsi ;
        if (e.locmngr_n = i) then
          insert agent a's element (e.agent_id, dest_n, e.agent_t+1, true, true)
            into AgentLocsi ;
        else {
          send a message m_initiated(e.agent_id) to node e.locmngr_n ;
          wait a message m_reply(e.agent_id) from node e.locmngr_n ;
        }
        send a message migrate(a, e) to node dest_n ;
        remove e from RunningAgentsi ;

Actions taken when Ni receives a message migrate(a, loc_e) from node previous_n
        loc_e.agent_t ← loc_e.agent_t + 1 ;
        if (loc_e.locmngr_n ≠ i) then
          if (agent a want node Ni to be its locator) then {
            prevlocmngr_n ← loc_e.locmngr_n ;
            loc_e.locmngr_n ← i ;
            send a message changelmngr(loc_e.agent_id, loc_e.agent_t)
              to node prevlocmngr_n ;
            if ((prevlocmngr_n ≠ a's home node) ∧ (a wants its home node to
                be informed of its new location manager)) then
              send a message update(loc_e.agent_id, loc_e.agent_t)
                to a's home node ;
          }else
            send a message update(loc_e.agent_id, loc_e.agent_t)
              to node loc_e.locmngr_n ;
        if (the element for agent a is in AgentLocsi) then
          remove agent a's element from AgentLocsi ;
        insert agent a's element loc_e into RunningAgentsi ;
        execute agent a ;

Actions taken when Ni receives a message m_initiated(a_id) from node running_n
        e ← look up agent a_id's element from AgentLocsi ;
        e.ismigrating_f ← true ;
        send a message m_reply(a_id) to node running_n ;

Actions taken when Ni receives a message update(a_id, timestamp)
  from node running_n
        e ← look up agent a_id's element from AgentLocsi ;
        e.running_n ← running_n ;
        e.agent_t ← timestamp ;
        if (a_id's home node ≠ i) then {
          e.ismigrating_f ← false ;
          forward all messages buffered in a_id's message queue
            to node e.running_n ;
        }

Actions taken when Ni receives a message changelmngr(a_id, timestamp)
  from node newlocmngr_n
        e ← look up agent a_id's element from AgentLocsi ;
        e.running_n ← newlocmngr_n ;
        e.agent_t ← timestamp ;
        e.ismanaging_f ← false ;
        e.ismigrating_f ← false ;
        forward all messages buffered in a_id's message queue to node e.running_n;
```

(그림 4) 에이전트 이주 시 서비스 노드 Ni를 위한 프로시저들

4.2 메시지 전달 서비스

앞에서 언급한 적응적 디렉토리 서비스 프로토콜에서 제공하는 이동 에이전트 위치정보의 효율성을 극대화시키기 위해서는 서비스 노드마다 <표 3>에 소개되는 스마트한 에이전트 위치캐쉬 ALocsCachei를 사용하여 메시지 전달 프로토콜을 설계한다. 제안하는 메시지 전달 프로토콜이 어떻게 이러한 목적을 달성할 수 있는지를 (그림 5)를 통해 살펴보도록 하자. 이 그림은 에이전트 a가 자신의 여정에 따라 여러 노드들을 방문하는 동안, 에이전트 b가 세 개의 메시지 m1, m2, m3를 순서대로 에이전트 a에게 송신하는 과정을 보여준다.

〈표 3〉 ALocsCachei

정의	노드 N_i에서 수행하는 에이전트들이 통신하는 각 이동 에이전트의 위치정보를 일시적으로 저장하기 위한 캐쉬	
필드 구성	• agent_id:	통신하는 이동 에이전트의 식별자
	• forward_n:	노드 N_i가 현재 알고 있는 에이전트 agent_id의 위치관리자 노드의 식별자
	• agent_t:	에이전트 agent_id의 위치정보가 노드 $N_{forward_n}$에서 관리되고 있을 때, 그 에이전트에 할당된 타임스탬프

(그림 5) (a)는 에이전트 a가 홈 노드 N_{home}로부터 노드 N_2로 이주한 후에, 노드 N_{sender}에서 수행하는 에이전트 b가 a에게 첫 번째 메시지인 m1을 송신하는 예이다. 이 경우, 노드 N_{sender}는 그림에서와 같이 현재 자신의 위치캐쉬 $ALocsCache_{sender}$에 a를 위한 위치정보가 전혀 존재하지 않는다. 따라서, N_{sender}는 a를 위한 원소인 (id_a, Home, 0)를 생성하여 자신의 캐쉬에 저장한다. 이후에, 그 노드는 메시지 m1을 a의 홈 노드 N_{home}로 송신한다. 이 때, 노드 N_{home}는 자신이 현재 위치 관리자가 아니기 때문에, a의 다음 포워더 N_1에게 메시지 m1을 전달한다. 이 경우, 노드 N_1은 현재 a의 위치 관리자이기 때문에, 자신이 직접 에이전트 a가 현재 수행하는 노드 N_2로 메시지 m1을 전달할 수 있다. 또한, 노드 N_{sender}가 a의 현재 위치 관리자의 식별자를 알지 못하기 때문에, 위치 관리자 N_1은 N_{sender}에게 a의 식별자(=id_a), 자신의 식별자(=1), 현재의 타임스탬프 값(=2)을 updateCache라는 메시지에 포함시켜 전달한다. 이 메시지를 수신한 N_{sender}는 그 메시지를 사용하여 자신의 캐쉬 $ALocsCache_{sender}$에 a의 원소를 갱신한다.

(그림 5) (b)은 에이전트 a가 노드 N_2로부터 노드 N_4로 이주한 후에, 노드 N_{sender}에서 수행하는 에이전트 b가 a에게 두 번째 메시지인 m2를 송신하는 예이다. 이 경우, 노드 N_{sender}는 그림에서와 같이 현재 자신의 위치캐쉬 $ALocsCache_{sender}$ 내에 a를 위한 원소 (id_a, 1, 2)를 참조하여, 메시지 m2을 a의 포워더 N_1로 송신한다. 이 때, 노드 N_1은 현재 a의 위치 관리자이기 때문에, 자신이 직접 에이전트 a가 현재 수행하는 노드 N_4로 메시지 m2를 전달할 수 있고, 노드 N_{sender}가 a의 현재 위치 관리자의 식별자를 알기 때문에, 위치 관리자 N_1은 N_{sender}에게 a의 위치정보를 알려줄 필요가 없다.

(그림 5) (c)는 에이전트 a가 노드 N_4로부터 다음 위치 관리자 N_5를 거쳐 노드 N_6로 이주한 후, 노드 N_{sender}에서 수행하는 에이전트 b가 a에게 세 번째 메시지 m3을 송신하는 예이다.

이 경우, 노드 N_{sender}는 그림에서와 같이 현재 자신의 위치캐쉬 $ALocsCache_{sender}$내에 a를 위한 원소 (id_a, 1, 2)를 참조하여, 메시지 m3를 a의 포워더 N_1로 송신한다. 이 때, 노드 N_1는 자신이 현재 위치 관리자가 아니기 때문에, a의 다음 포워더 N_5에게 메시지 m3를 전달한다. 이 경우, 노드 N_5는 현재 a의 위치 관리자이기 때문에, 자신이 직접 에이전트 a가 현재 수행하는 노드 N_6에게 메시지 m3를 전달할 수 있다. 또한, 노드 N_{sender}가 a의 현재 위치 관리자의 식별자를 알지 못하기 때문에, 위치 관리자 N_5는 N_{sender}에게 a의 최신 위치정보(id_a, 5, 6)를 updateCache 메시지에 포함시켜 전달한다. 이 메시지를 수신한 N_{sender}는 그 메시지를 사용하여 자신의 캐쉬 $ALocsCache_{sender}$에 a의 원소를 갱신한다.

메시지 전달 프로토콜에서 각 서비스 노드를 위한 프로시저들은 (그림 6)에서 정형적으로 표현된다.

5. 성능 평가

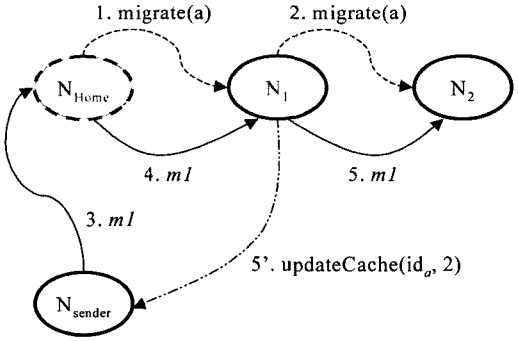
5.1 시뮬레이션 환경

이 절에서는 본 논문에서 설계한 에이전트간 통신 프로토콜(Ours)과 기존의 포워딩 포인터 기반 프로토콜(Forwarding)과의 성능을 분석하기 위해 PARSEC 이산-이벤트 시뮬레이션 언어[5]를 사용하여 대규모 시뮬레이션을 수행하고자 한다. 이 시뮬레이션에서의 성능평가 두 가지 기준은 다음과 같다: N_{Hops}-한 메시지가 수신 에이전트로 전달되기 위해 포워딩되는 노드의 수, $N_{Entries}$-각 노드에서 유지해야하는 에이전트 위치정보 엔트리 수. 시뮬레이션되는 시스템의 구성은 다음과 같다. 시뮬레이션되는 네트워크의 구성은 <표 4>와 같이 전체 서비스 노드의 수는 100이고 각 노드는 좌표(x, y)에 위치하고 있다. 또한, 모든 노드들은 서로 상호 연결되어 있다.

인접한 임의의 두 노드들은 대여폭이 10Mbps, 전파지연시간이 2ms인 랜(LAN) 환경 내에 연결되어 있다고 가정한다. 그리고, 가장 큰 거리를 가진 두 노드 즉, 노드 (1,1)과 노드 (10,10)는 대여폭 64kbps와 전파지연시간 100ms인 네트워크로 연결되어 있다고 한다. 이러한 네트워크 환경을 간단하게 하기 위해, 임의의 두 노드간의 대여폭과 전파지연시간이 그들간의 거리에 따라 선형적으로 분산된다고 한다.

본 시뮬레이션 환경에서 에이전트간 통신 프로토콜의 시뮬레이션 파라미터들로는 다음과 같다: $S_{senders}$-네트워크 내에 에이전트 a로 메시지를 송신할 수 있는 송신자 집합, $T_{threshold}$-에이전트 a가 자신의 위치 관리자를 변경할 것인지를 판단하기 위한 임계 값. 먼저, 본 시뮬레이션에서는 하나의 수신 이동 에이전트 a가 <표 4>에서의 노드 (1,1)부터 노드 (10,10)까지 순차적으로 이주한다. 또한, 에이전트 a는 이주 전에 임의로 발생된 확률 p의 값이 $T_{threshold}$보다 높으면 다음

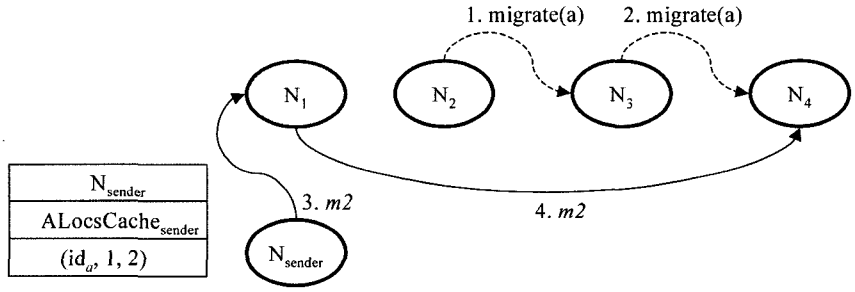
N_{Home}	N_1	N_2
$AgentLocs_{Home}$	$AgentLocs_1$	$RunningAgents_2$
(id_a, 1, 1, false, false)	(id_a, 2, 2, true, false)	(id_a, 1, 2)



	N_{sender}
	$ALocsCache_{sender}$
Step 1)	(id_a, Home, 0)
Step 2)	(id_a, 1, 2)

(a)

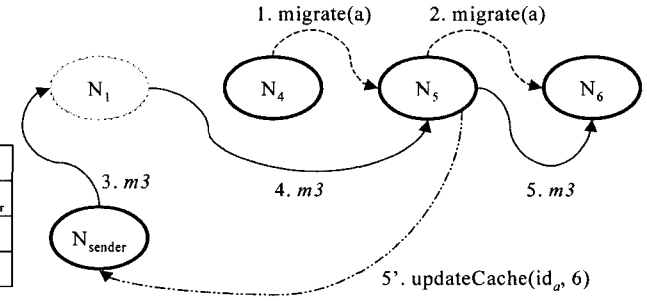
	N_1	N_3	N_4
	$AgentLocs_1$	$RunningAgents_2$	$RunningAgents_3$
Step 1)	(id_a, 3, 3, true, false)	(id_a, 1, 3)	
Step 2)	(id_a, 3, 3, true, true)	(id_a, 1, 3)	
Step 3)	(id_a, 4, 4, true, false)		(id_a, 1, 4)



N_{sender}
$ALocsCache_{sender}$
(id_a, 1, 2)

(b)

	N_1	N_5		N_6
	$AgentLocs_1$	$RunningAgents_5$	$AgentLocs_5$	$RunningAgents_6$
Step 1)	(id_a, 5, 5, false, false)	(id_a, 5, 5)		
Step 2)	(id_a, 5, 5, false, false)		(id_a, 6, 6, true, true)	
Step 3)	(id_a, 5, 5, false, false)		(id_a, 6, 6, true, false)	(id_a, 5, 6)



	N_{sender}
	$ALocsCache_{sender}$
Step 1)	(id_a, 1, 2)
Step 2)	(id_a, 5, 6)

(c)

(그림 5) 에이전트 a가 자신의 여정에 따라 여러 노드들을 방문하는 동안, 에이전트 b가 세 개의 메시지 m1, m2, m3를 순서대로 에이전트 a에게 송신하는 예

```
Actions taken when a message msg is sent to agent a_id
        if (there is the agent a_id's element e in RunningAgents_i)  then
          deliver msg to agent a_id ;
        else if (there is the agent a_id's element e in ALocsCache_i)  then
          next_n ← e.forward_n ;
        else {
          next_n ← the identifier of agent a_id's home node ;
          insert agent a_id's element (a_id, next_n, 0) into ALocsCache_i ;
        }
        send a message msg with (a_id, N_i) to node next_n ;

Actions taken when N_i receives a message msg with (a_id,msgsource_n) from node msgforwarding_n
        if (there is the agent a_id's element e in RunningAgents_i) then
          deliver msg to agent a_id ;
        else if (there is the agent a_id's element e in AgentLocs_i) then
          if (e.ismanaging_f = true) then
            if (e.ismigrating_f = true) then
              insert the message msg with (a_id, msgsource_n) into agent a_id's message queue ;
            else
              send a message msg with (a_id, msgsource_n) to node e.running_n ;
          else
            send a message forwarding(a_id,msg,msgsource_n) to node e.running_n ;

Actions taken when N_i receives a message forwarding(a_id, msg, msgsource_n)
  from node msgforwarding_n
        if (there is the agent a_id's element e in RunningAgents_i)  then {
          deliver msg to agent a_id ;
          send a message updatecache(a_id, e.agent_t) to node msgsource_n ;
        } else if (there is the agent a_id's element e in AgentLocs_i) then
          if (e.ismanaging_f = true) then {
            if (e.ismigrating_f = true) then
              insert the message msg with (a_id, msgsource_n) into agent a_id's message queue ;
            else
              send a message msg with (a_id, msgsource_n) to node e.running_n ;
            send a message updatecache(a_id, e.agent_t) to node msgsource_n ;
          } else
            send a message forwarding(a_id,msg,msgsource_n) to node e.running_n ;

Actions taken when N_i receives a message updatecache(a_id, timestamp) from node newlocmngr_n
        e ← look up agent a_id's element from ALocsCache_i ;
        if (e.agent_t < timestamp) then {
          e.forward_n ← newlocmngr_n ;
          e.agent_t ← timestamp ;
        }
```

(그림 6) 메시지 전달 시 서비스 노드 Ni를 위한 프로시저들

〈표 4〉 시뮬레이션되는 네트워크 위상

(1,10)	(2,10)	(3,10)	(4,10)	(5,10)	(6,10)	(7,10)	(8,10)	(9,10)	(10,10)
(1, 9)	(2, 9)	(3, 9)	(4, 9)	(5, 9)	(6, 9)	(7, 9)	(8, 9)	(9, 9)	(10, 9)
(1, 8)	(2, 8)	(3, 8)	(4, 8)	(5, 8)	(6, 8)	(7, 8)	(8, 8)	(9, 8)	(10, 8)
(1, 7)	(2, 7)	(3, 7)	(4, 7)	(5, 7)	(6, 7)	(7, 7)	(8, 7)	(9, 7)	(10, 7)
(1, 6)	(2, 6)	(3, 6)	(4, 6)	(5, 6)	(6, 6)	(7, 6)	(8, 6)	(9, 6)	(10, 6)
(1, 5)	(2, 5)	(3, 5)	(4, 5)	(5, 5)	(6, 5)	(7, 5)	(8, 5)	(9, 5)	(10, 5)
(1, 4)	(2, 4)	(3, 4)	(4, 4)	(5, 4)	(6, 4)	(7, 4)	(8, 4)	(9, 4)	(10, 4)
(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)	(6, 3)	(7, 3)	(8, 3)	(9, 3)	(10, 3)
(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)	(6, 2)	(7, 2)	(8, 2)	(9, 2)	(10, 2)
(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)	(6, 1)	(7, 1)	(8, 1)	(9, 1)	(10, 1)

〈표 5〉 성능 파라미터

파라미터	설명
$T_{residence}$	각 에이전트가 한 노드에서 부분적인 작업을 하기 위해 머무르는 시간
N_{Agents}	이동 에이전트 시스템에서 생성되는 에이전트 수

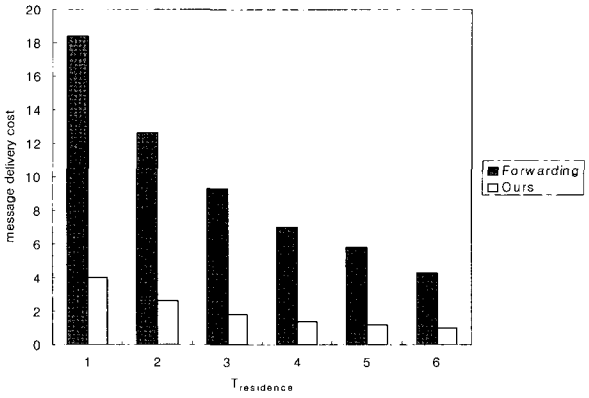
노드가 a의 위치 관리자가 된다. 이러한 파라미터 $T_{threshold}$은 a의 생명주기 동안 그 값이 변하지 않는다고 가정한다. 집합 $S_{senders}$내의 송신자 에이전트들은 시스템내에서 임의적으로 분산되어 있다. 이들은 이주할 때마다 100개의 노드들 중 자신의 목적 노드들을 선택한다. 또한, 에이전트들간 데이터 메시지의 크기는 512bytes라 하고, 에이전트간 통신 프로토콜을 위한 제어 메시지의 크기는 128bytes라 한다. 마지막으로 본 시뮬레이션에서 성능측정을 위한 주요한 파라미터들은 <표 5>와 같다.

5.2 시뮬레이션 결과 및 분석

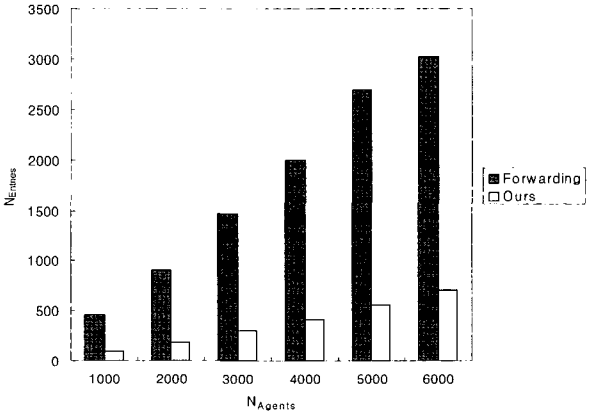
(그림 7)은 각 에이전트가 한 노드에서 부분적인 작업을 하기 위해 머무르는 시간 $T_{residence}$를 일정한 범위로 변화시키면서 두 개의 에이전트간 통신 프로토콜 Ours와 Forwarding의 평균 메시지 전달 비용 N_{Hops}을 보여준다. (그림 7)에서 두 프로토콜의 $T_{residence}$가 적을수록, 그들의 메시지 전달 비용이 높아진다. 이는 각 에이전트가 한 노드에서 머무르는 시간이 적음에 따라, 그 프로토콜들은 각 에이전트 위치관리를 위한 포워딩 포인터의 수가 증가하기 때문이다. 그러나, Ours의 N_{Hops}는 Forwarding보다 매우 낮음을 알 수 있다. 특히, $T_{residence}$가 적을수록, 그들의 메시지 전달 비용간의 차이가 더 큰 것을 알 수 있다. 이 경우, Ours의 N_{Hops}는 Forwarding의 N_{Hops}에 비해 76%~80%만큼 줄인다.

(그림 8)은 본 시뮬레이션 환경에서 생성되는 이동 에이전트 수 N_{Agents}를 일정한 범위로 변화시키면서 두 프로토콜들의 각 노드에서 유지해야하는 에이전트 위치정보 엔트리 수 $N_{Entries}$를 보여준다. (그림 8)에서 두 프로토콜의 이동 에이전트 수 N_{Agents}가 증가할수록, 그들의 $N_{Entries}$가 높아진다. 이는 시스템내에서 생성되는 에이전트 수가 높음에 따라, 그 프로토콜들은 각 노드에서 유지해야하는 에이전트들의 위치정보량이 증가하기 때문이다. 그러나, Ours의 $N_{Entries}$는 Forwarding보다 매우 적음을 알 수 있다. 특히, 이동 에이전트 수 N_{Agents}가 증가할수록, 두 프로토콜들에서 각 노드가 유지해야하는 평균 위치정보량의 차이가 보다 더 커짐을 알 수 있다. 이 경우, Ours의 $N_{Entries}$는 Forwarding의 $N_{Entries}$에 비해 76%~79%만큼 줄인다.

따라서, 이러한 시뮬레이션 결과로부터 본 연구에서 제안된 에이전트간 통신 프로토콜 Ours는 Forwarding에 비해 이동 에이전트 시스템 규모가 매우 커지더라도 적은 메시지 전달 비용을 발생시키고, 각 서비스 노드에서 이동 에이전트들의 위치정보를 유지하는데 필요로 하는 저장소 크기를 매우 줄임으로써 시스템의 확장성을 매우 향상시킬 수 있다는 결론을 내릴 수 있다.



(그림 7) 메시지 전달 비용 vs. $T_{residence}$



(그림 8) $N_{Entries}$ vs. N_{Agents}

6. 결 론

본 논문에서는 홈 노드에 대한 의존성을 줄이면서 각 서비스 노드에 의해 유지되는 에이전트 위치정보량과 각 메시지의 전달 시간을 매우 감소시키는 적응적 에이전트 통신 프로토콜을 제안하였다. 이러한 목적을 달성하기 위해, 제안된 프로토콜은 단지 각 이동 에이전트의 방문노드들 중 일부분만을 포워더가 되도록 한다. 이러한 특성은 각 에이전트가 이주할 때마다, 자신의 위치를 현재 위치관리자에게만 갱신하도록 함으로써 가질 수 있다. 그러나, 이 프로토콜은 기존의 포워딩 포인터 기반 기법에 비해 에이전트 이주마다 발생하는 위치갱신 비용이 조금 더 높을 수 있다. 그러나, 각 에이전트가 여러 가지 성능 요소들을 알맞게 고려함으로써 자신의 방문노드들 중 일부를 포워더로 정한다면, 이러한 비용의 차이는 거의 무시할 수 있을 정도로 매우 적다.

또한, 제안된 프로토콜은 각 서비스 노드마다 스마트한 에이전트 위치캐쉬를 사용하여 이동 에이전트 위치정보의 효율성을 극대화시킨다. 즉, 에이전트의 위치관리자 식별자를 서비스 노드의 에이전트 위치캐쉬에 유지하게 함으로써, 기존 기법에 비해 노드의 캐쉬 갱신 횟수를 매우 줄일 수 있다. 따라서, 이 프로토콜은 본 논문의 시뮬레이션 결과에서 알 수 있는 것처럼 홈 노드에 대한 의존성을 줄이면서 각 서비스 노드에 의해 유지되는 에이전트 위치정보량과 각 메시지의 전달 시간을 매우 감소시킨다.

본 논문에서 제안한 에이전트간 통신 프로토콜은 어떤 특정한 이동 에이전트 시스템 환경에 대한 적용에 초점을 맞추고 있지 않기 때문에, 이동 에이전트가 자율적으로 위치관리자를 선택하는 구체적인 방법은 제시되지 않고 있다. 그러나, 각 에이전트 시스템의 구체적인 특성, 예를 들면, 서비스 노드의 보안정책 및 네트워크 위상과 지연시간, 에이전트간 통신패턴, 에이전트 여정 등이 위치관리자 선택 시 해당 에이전트 시스템의 전체 성능에 영향을 미칠 수 있는 중요한 선택기준이 될 수 있다. 따라서, 향후 연구과제로 앞에서 언급한 다양한 시스템 환경의 특성에 따라 제안된 프로토콜의 확장성 및 효율성에 대한 보다 다각적인 실험을 통한 분석을 수행하고자 한다.

참 고 문 헌

[1] J. Ahn, "An Adaptive Communication Mechanism for Highly Mobile Agents," Lecture Notes In Computer Science (ICCS04), Springer-Verlag, Vol.3036, pp.192-199, 2004.

[2] F. Baschieri, P. Bellavista and A. Corradi, "Mobile Agents for Qos Tailoring, Control and Adaptation over the Internet: The UbiQoS Video on Demand Serbvice," *In Proc. of the 2nd International Symposium on Applications and the Internet*, pp.109-118, 2002.

[3] P. Bellavista, A. Corradi, and C. Stefanelli, "The Ubiquitous Provisioning of Internet Services to Portable Devices," *IEEE Pervasive Computing*, Vol.1, No.3, pp.81-87, 2002.

[4] J. Baumann, F. Hohl, K. Rothermel and M. Strasser, "Mole-Concepts of a Mobile Agent System", *World Wide Web*, Vol.1, No.3, pp.123-137, 1998.

[5] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin and H. Y. Song, "Parsec: A Parallel Simulation Environments for Complex Systems," *IEEE Computer*, pp.77-85, 1998.

[6] W. Belle, K. Verelst and T. D. Hondt, "Location transparent routing in mobile agent systems merging name lookups with routing," *In Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems*, pp.207-212, 1999.

[7] J. Cao, X. Feng, J. Lu and S. Das, "Mailbox-based scheme for mobile agent communications," *IEEE Computer*, Vol.35, No.9, pp.54-60, 2002.

[8] J. Desbiens, M. Lavoie and F. Renaud, "Communication and tracking infrastructure of a mobile agent system," *In Proc. of the 31st Hawaii International Conference on System Sciences*, Vol.7, pp.54-63, 1998.

[9] A. Fuggetta, G.P.Picco and G. Vigna, G., "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, Vol.24, No.5, pp.342-361, 1998.

[10] M. Fukuda, Y. Tanaka, N. Suzuki, L.F. Bic and S. Kobayashi, "A Mobile-Agent-Based PC Grid," *In Proc. of the Fifth Annual International Workshop on Active Middleware Services*, pp.142-150, 2003.

[11] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[12] L. Moreau, "Distributed Directory Service and Message Router for Mobile Agents." *Science of Computer Programming*, Vol.39, No.2-3, pp.249-272, 2001.

[13] L. Moreau, "A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers," *In Proc. of The 17th ACM Symposium on Applied Computing*, pp.93-100, 2002.

[14] A. L. Murphy and G. P. Picco, "Reliable Communication for Highly Mobile Agents," *Journal of Autonomous Agents and Multi-Agent Systems*, Vol.5, No.1, pp.81-100, 2002.

[15] L. Moreau and D. Ribbens, "Mobile Objects in Java", *Scientific Programming*, Vol.10 , No.1, pp.91-100, 2002.

[16] ObjectSpace, Voyager, *http://www.objectspace.com/*.

[17] S. Pleisch and A. Schiper, "Fault-Tolerant Mobile Agent Execution", *IEEE Transactions on Computers*, Vol.52, No.2, pp.209-222, 2003.

[18] M. Ranganathan, M. Bednarek and D. Montgomery, "A Reliable Message Delivery Protocol for Mobile Agents," *In Proc. of the 2nd International Symposium on Agent Systems and Applications and the 4th International Symposium on Mobile Agents*, LNCS 1882, pp.206-220, 2000.

[19] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young and B. Peet, "Concordia: An Infrastructure for Collaborating Mobile Agents", *Lecture Notes In Computer Science*, Vol. 1219, 1997.

안 진 호

e-mail : jhahn@kyonggi.ac.kr

1997년 고려대학교 컴퓨터학과(이학학사)

1999년 고려대학교 컴퓨터학과(이학석사)

2003년 고려대학교 컴퓨터학과(이학박사)

2003년~현재 경기대학교 정보과학부 전자계산학과 조교수

관심분야 : 결함포용 분산시스템, 이동에이전트 시스템, 그룹통신, p2p 컴퓨팅