효율적인 통신망 관리를 위한 이동에이전트 응용

전병국*, 김영철**
*원주대학 사무자동화과
**LG산전 중앙연구소
e-mail : jeonbk@sky.wonju.ac.kr
robertkim@mail.lgis.co.kr

An Application of Mobile Agents for Efficient Network Management

Byung-Kook Jeon*, Young-Chul Kim**
*Dept of Office Automation, Wonju Nat'l College
**LGIS Laboratory

요약

본 논문은 자바 환경의 이동에이전트 시스템인 JAMAS를 기반으로 한 이동에이전트를 응용하여 망 관리를 구현한다. 오늘날의 통신망은 전형적으로 이형의 다중 플랫폼이며, 서로 다른 제조회사의 통신망 장비를 복합적으로 구성하여 사용하기 때문에 이식성과 플랫폼 독립성을 지원하는 이동에이전트를 이용하는 것이 효율적이다. 따라서, 이동에이전트는 관리 장비에 SNMP 에이전트처럼 정적으로 존재하는 것이 아니라, 관리자를 대신하여 이동하고 자율적인 실행을 통해 통신망 장애를 진단하고 고칠 수 있도록 독립적으로 기능 구현이 가능하며, 나아가 지능이 있는 이동에이전트를 제공할 수 있다. 본 논문은 통신망 관리를 위해서 SNMP의 관리 정보인 MIB를 이동에이전트가 접근하여 수집한 후, 관리자에게 전달하는 MIB 브라우저 역할 수행의 이동에이전트를 구현하고, 성능 평가를 하여 효율성을 입증한다

1. 서론

오늘날 LAN이나 WAN, 인터넷 등 규모가 큰 통신망 환경에서는 다수의 컴퓨터 시스템이나 통신망 장비 제공자가 제공하는 MIB(Management Information Base) 관리 정보의 다양함을 지원 및 운영할 수 있는 관리자 시스템이 필요하다. 더욱이, 제조 회사마다 지원하는 MIB 관리 정보는 통신망 관리자들을 점점 더 힘들게 하고 있으며, 인터넷의 발전으로 인해 통신망 관리자들은 손쉽게 웹을 이용해 관리할 수 있는 편리성을 추구하므로 크고 통합적인 기존의 관리자 응용 시스템인 NMS(Network Management System)들은 더 이상 적합하지 않다[1,2,3,7]. 또한, 기존의 NMS들은 분산도가 약한 중앙 집중형으로 운영되고 있는 클라이언트/서버 모델을 기반으로 한 분산 시스템으로서, 클라이언트 역할을 하는 관리자로부터 요청을 기다리고 응답을 전달한다. 이러한 접근 방법은 망 관리자 측면에서 정보 병목(bottleneck) 현상이 자주 발생한다. 뿐만 아니라 단지 망 관리를 위한 폴링(polling) 오버헤드, 저수준의 보안 문제, 이기종간의 호환성 문제 등으로 인해 통신 트래픽 효율을 저하시킨다[4,7].

이같은 문제 해결을 위해 본 논문에서는 이미 개발된 이동에이전트 시스템인 JAMAS(JAva Mobile Agent System)를 기반으로 이동에이전트를 응용하여 망 관리를 구현한다[10,11]. 이동에이전트를 이용하는 가장 큰 이유로서, 오늘날의 통신망은 전형적으로 이형의 다중 플랫폼이며, 서로 다른 제조회사의 통신망 장비를 복합적으로 구성하여 사용하기 때문에 이식성과 플랫폼 독립성을 지원하는 이동에이전트를 이용하는 것이 효율적이다[3,5]. 또한, 이동에이전트는 관리 장비에 SNMP(Simple Network Management Protocol) 에이전트처럼 정적으로 존재하는 것이 아니라, 관리자를 대신하여 이동하고 자율적인 실행을 통해 통신망 장애를 진단하고 고칠 수 있도록 독립적으로 기능 구현이 가능하며, 나아가 지능이 있는 이동에이전트를 제공할 수 있다[9,11]. 따라서, 본 논문에서는 통신망 관리를 구현하기 위한 방법으로써 JAMAS 기반 이동에이전트를 활용한다. 즉, SNMP 에이전트가 관리하는 MIB 정보를 접근 및 수집하여 제공하는 MIB 브라우저 역할의 이동에이전트를 구현하며, 성능 평가를 한다.

본 논문의 구성은 2장에서는 통신망 관리 구현을 위해 MIB 브라우저 기능의 이동에이전트 기능과 해당 이동에이전트를 수용할 수 있는 플랫폼으로서 JAMAS에 대해 분석하고, 3장에서는 JAMAS가 제공하는 기능에 따른 2가지 모델을 각각 구현한다. 그리고 4장에서는 성능 분석을 통하여 2가지 구현 모델에 대해 비교 평가한다. 끝으로 5장에서는 결론 및 앞으로의 연구 방향을 제시한다.

2. MIB 접근을 위한 이동에이전트와 JAMAS

MIB 접근은 기존 방식처럼 SNMP 에이전트와 대화를 하거나 SNMP 에이전트를 새로 개발할 수 있다. 이동에이전트를 이용하여 새로 SNMP 에이전트 및 모든 기능을 만족시키는 연구[4,5]도 있지만, 본 논문에서는 기존 시스템에 있는 SNMP 에이전트를 직접적으로 변경하지 않고 SNMP 에이전트와 대화를 통한 MIB 접근 역할에 적합하도록 JAMAS 기반의 이동에이전트를 응용하여 적용한다. 응용된 이동에이전트일 지라도 기본적으로는 이동에이전트의 특성을 모두 만족한다[11]. 임의의 JAMAS 기반 호스트에서 이주된 이동에이전트는 언마샬(un-marshal)되고, 인스턴스 혹은 이전 상태 정보에 대해 재인스턴스되어 'begin' 메소드를 호출하여 자율적으로 수행하는 쓰레드이다. 이 쓰레드는 지역 호스트의 SNMP 에이전트간의 통신을 통해서 SNMP 에이전트가 제공하는 MIB 정보를 접근하여 관리자에게 즉시 혹은 수집하여 나중에 전달할 수 있다. 해당 에이전트는 이후 방문할 다른 곳이 있을 경우 이같은 방법으로 이주를 계속 수행한다.

JAMAS는 동형과 이형 환경하에서 자바 가상 머신 환경을 기반으로 구현된다[8]. 연동된 통신망에서 JAMAS는 각 서버들과 PC 시스템들에 탑재되어 있고, 각 시스템들은 하나 이상의 JAMAS 환경을 지원한다. 이때 JAMAS는 상호 독립적인 환경으로 처리된다. 즉, JAMAS는 자바(Java) 이동에이전트를 대상으로 입/출 기능이 있는 추상적인 장소를 제공하는 환경이며, 한 호스트에 다수의 독립적인 JAMAS가 존재할 수 있다. 기타 JAMAS의 구조와 기능은 [11]을 참조한다. 본 논문에서는 PC나 NT, 워크스테이션, 라우터 등을 NE(Network Element)라 하고, 각각의 NE에는 자바 가상 머신이 탑재되어 있다고 가정할 때, 탑재된 NE들은 자신의 운영체제와 함께 JAMAS가 구동하고 있으며, 따라서 이동에이전트는 언제든지 사용자 혹은 관리자의 요구에 따라 해당 NE들로 이동할 수 있다. 그러나, 통신망 시스템이나 라우터, LAN 스위치 같은 통신 장비들은 각 제조 회사별로 고유의 SNMP 에이전트를 지원하고 있으며, 동시에 자바 가상 머신이 탑재되어 있지 않기 때문에 이들 장비에 관련된 MIB 정보 수집은 클라이언트/서버처럼 원격 접속을 통해 이루어진다. 가까운 장래에 자바 가상 머신 환경을 제공하는 자바 칩(chip)이나 Jini를 사용할 경우에는 본 본문의 JAMAS가 실행되어 이동에이전트를 이주시킬 수 있는 동일한 환경이 될 수 있으며, 현재 통신망 장비 제조 회사의 일부 제품은 이미 자바 가상 머신이 탑재 혹은 예정중이다[2,6].

그러므로, 각 NE마다 자바 가상 머신 내부에는 JAMAS들이 실행되고 있다. 각 JAMAS들은 통신망을 통해 이주되어 오는 인증된 이동에이전트를 받고, 인스턴스하며, 제어를 하는 등 에이전트 실행을 지원한다. 인스턴스된 이동에이전트는 SNMP 에이전트와 대화를 통해서 지역 NE의 MIB 정보를 접근한다. 혹은 지역 NE의 MIB 정보를 접근할 수 없을 때에는 원격지의 SNMP 에이전트가 제공하는 NE의 MIB 정보도 접근한다. 즉, 통신망 프린터나 라우터, ATM 장비 등에 자바 가상 머신이 설치되어 있지 않기 때문에, 이동에이전트는 원격지인 워크스테이션이나 NT에서도 통신망 프린터 같은 NE의 MIB 정보를 접근함을 뜻한다. 그리고, SNMP 에이전트는 자바로 구현하였을 경우 가상 머신 환경에 놓일 수도 있으며, 혹은 그림 1처럼 자바 가상 머신과 무관하게 시스템 독립적인 실행 영역에서 데몬(deamon) 프로세스로서 수행한다. 이같은 방식의 JAMAS와 이동에이전트간의 상관 관계는 그림 1과 같다. 그림 1은 이동에이전트가 관리자에 의해 이주 명령을 받았을 때, 각 NE에서 수행하는 것을 보여주고 있다. 각 NE는 내부에서 프로세스들을 수행하는 시스템이며, 점선의 원들과 JAMAS는 NE에서의 상주 프로세스이며, 원은 쓰레드(thread)이다. 또한, 하단 실선은 NE의 응용 프로그램 실행 영역과 커널 계층(kernel layer)을 구분한 것이다. 여기서 하드웨어 질의 프로세스는 하드웨어에 관한 정보를 가져오기 위한 질의를 수행하거나 구성 정보를 보내는 프로세스(process)이다. 아울러, 그림 1에서의 곡선은 관리자로부터 임무를 부여받은 자바 이동에이전트가 JAMAS 지원하에 이주되어 임무를 수행하고 다른 NE로 이주하는 시나리오를 보여준다. 그리고 곡선의 한쪽 끝이 화살표인데 이는 통신망의 어느 곳에서나 종료될 수 있다는 것을 뜻한다.

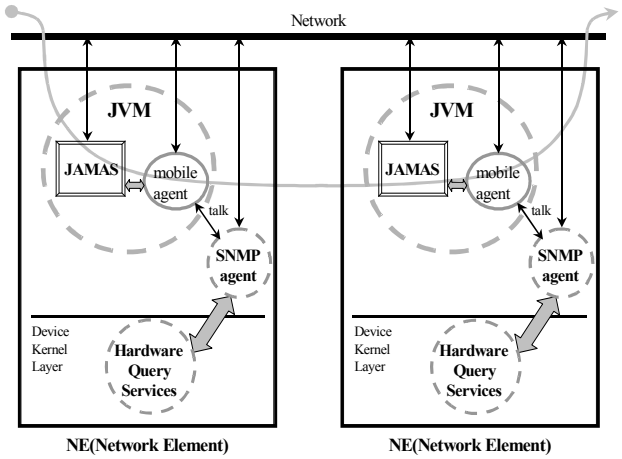


그림 1. NE들에서의 JAMAS와 이동에이전트

3. 구현

본 논문의 구현을 위해 MIB 브라우저 역할을 수행하도록 이동에이전트 응용 모델을 가정한다. 이동에이전트는 MIB 정보 접근을 위해서 계층 구조적인 객체 식별자(OID : Object Identification)를 가져야 한다. 객체 식별자 OID는 ANS.1 표기법에 따라 연속한 숫자 데이터를 사용한다. 또한, 관리자의 명령을 수행할 이동에이전트는 정보 수집 대상으로 각 지역 NE 혹은 원격지 NE들의 MIB 정보를 가져올 수 있어야 한다. 이를 위한 실험 환경은 통신망으로 연동된 한 개 이상의 NE를 대상으로 적용하며, 이에 대한 시스템과 각 주소는 표 1과 같다. 여기서 이동에이전트가 JAMAS 환경에 들어가기 위해서, 각 IP 주소는 개별 시스템 주소에 해당되며, 한 개의 IP 주소에 다수의 포트 번호는 해당 시스템에 다수의 JAMAS가 동작중임을 의미한다.

표 1 각 NE의 주소와 운영체제

NE#	NE의 IP 주소	Port #	운영체제	CPU
NE*a*	172.16.51.99	11070	Unix	Ultrasparc 300Mhz
NE*b*	172.16.53.20	4000	Windows 98	P-II 333Mhz
NE*c*	172.16.53.100	11010	Unix	Ultrasparc 300Mhz
NE*d*	172.16.53.70	9090	Windows NT	P-II 450Mhz (Dual CPU)
NE*e*	172.16.53.100	11020	Unix	Ultrasparc 300Mhz
NE*f*	172.16.53.10		Vendor-OS	
NE*g*	172.16.1.15		Vendor-OS	
NE*h*	172.16.53.21	5050	Windows 98	P-II 350Mhz

그러므로, 이동에이전트가 표 1의 각 NE들을 개별적으로 방문하여 처리하는 작업 수행함을 가정하였을 때, 본 논문에서는 다음의 두 방식에 의한 각각의 구현 모델과 실행 과정을 보인다.

• 구현 1 모델 : 단일 이동에이전트를 이용하여 5개의 NE들을 각기 방문할 때 지역 및 원격지 NE들에 대한 MIB 정보를 가져오는 것을 보인다.
• 구현 2 모델 : N개의 각 NE들에 대해 그룹 협력 작업을 수행하도록 N-1개의 작업에이전트와 주에이전트 간의 메시지 전달 수행이 이루어지는 것을 보인다.

3.1 구현 1 모델

첫 번째 구현 모델에서는 그림 2같은 라우팅 스케쥴에 따라 이주하는 단일 이동에이전트를 구현한다. 여기서, NE*a*, NE*b*, NE*e*는 지역 SNMP 에이전트와 대화하여 MIB 정보를 접근하는 경우와 NE*c*와 NE*d*는 원격지 NE들에 대한 MIB 시스템 정보를 가져오는 사례이다. 왜냐하면, 2장에서 기술한 것처럼 통신망 프린터나 통신망 장치에 자바 가상 머신이 없기 때문에 JAMAS가 이식되지 않았을 뿐 아니라, 실험에 적용한 이동에이전트 자체도 통신망 프린터에 이주할 수 없기 때문이다. 따라서, 이동에이전트는 지역뿐만 아니라 원격지의 다른 NE들인 통신망 프린터나 LAN 장비들의 SNMP 에이전트와 통신을 수행한 경우이다.

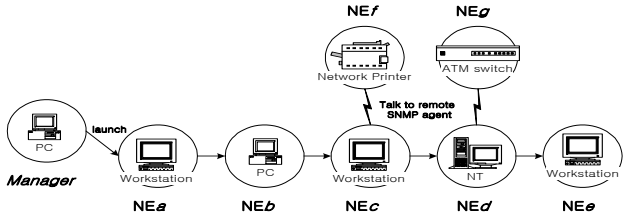


그림 2. 이동에이전트의 라우팅 스케쥴

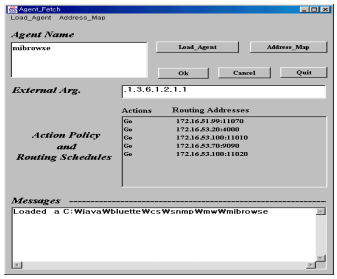


그림 3. 에이전트 적재 화면

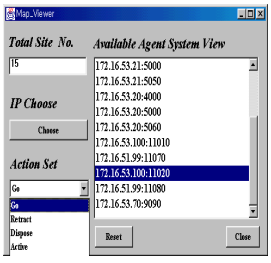
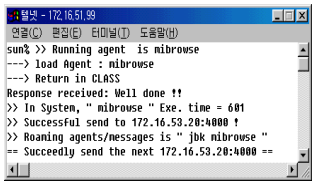
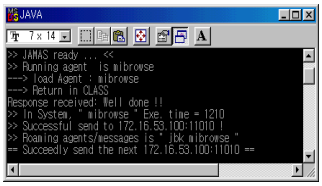


그림 4. Map_Viewer

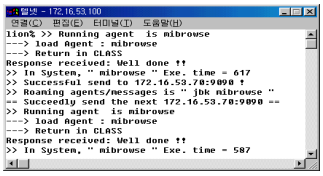
그림 2처럼 이동에이전트가 각 시스템 정보를 가져오기 위한 작업을 수행하려면 관리자가 JAMAS 사용자 인터페이스를 이용하여 자바 클래스 객체를 적재해야 한다. 이에 관한 그림 3은 MIB 정보를 수집해오는 에이전트를 적재한 화면이다. 이때, 외부 입력 데이터가 있으면 이를 인수로서 에이전트는 받아들인다. 또한, 적재된 에이전트에 대해 적절한 라우팅 정책을 사용자는 정의해야 한다. 왜냐하면 인터넷같은 환경에서 에이전트를 무작위로 이동시킬 수 없기 때문이다. 따라서 관리자는 이동에이전트를 위한 라우팅 정책을 설정하기 위해 그림 4의 Map_Viewer 창을 이용한다. 이 창의 'Available Agent System View'는 통신망에 연동된 각 NE마다 JAMAS가 동작중인지, 그리고 동작중이면 해당 NE 주소를 알려주는 에이전트 프록시(proxy)이다. 프록시의 이러한 기능은 이동에이전트 이주를 위한 목적지의 JAMAS에 대한 지역 투명성을 제공한다. 또한, 'Action Set'은 에이전트가 해당 호스트로 이동하였을 때 수행하여야 될 명령 기능들이다. 따라서 그림 3에 의하면 이동에이전트인 'mibrowse'는 1번씩 각 NE를 차례로 방문한다. 그림 3에 나타난 바와 같이 MIB 정보를 수집하기 위해 이동에이전트로서 'mibrowse' 클래스 객체가 선택되었으며 4개의 호스트로 이동하여 명령을 수행한다. 아울러 수집하려는 MIB 정보중에서 각 호스트 또는 NE의 시스템 정보만을 접근하기 위해 외부 인수인 객체 식별자로서 "1.3.6.1.2.1.1"로 부여하였다. 이후 초기 화면에서 'Go' 버튼을 누른다.



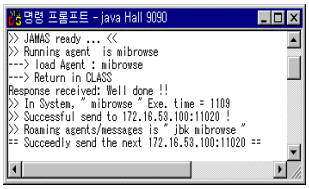
(a) NE*a* 실행 화면



(b) NE*b* 실행 화면



(c) NE*c*, NE*e* 실행 화면



(d) NE*d* 실행 화면

그림 5. 각 NE에서의 이동에이전트 실행

이동에이전트의 실제 실행됨은 그림 5처럼 각 지역 NE들에서 수행한 것을 화면 캡쳐한 것이다. 즉, 그림 5(a), (b), (c), (d)들은 이동에이전트가 각 호스트마다 사용자의 간섭없이 자율적으로 이동이 되었는지, 그리고 지역 시스템 내에서의 수행 시간은 얼마나 되는지를 확인하기 위해 지역 NE에서의 수행 화면을 보여준 것이다. 여기서, NE*c*와 NE*e*는 같은 시스템으로서, JAMAS가 하나 이상 존재할 수 있음과 동시에 라우팅 스케쥴에 따른 이동에이전트가 두 번 방문하였다. 따라서, 그림 5(a), (b), (c), (d) 화면에 메시지가 나타난 것처럼 에이전트는 각 시스템마다 올바르게 이주되어 성공적으로 처리되었음을 보여준다.

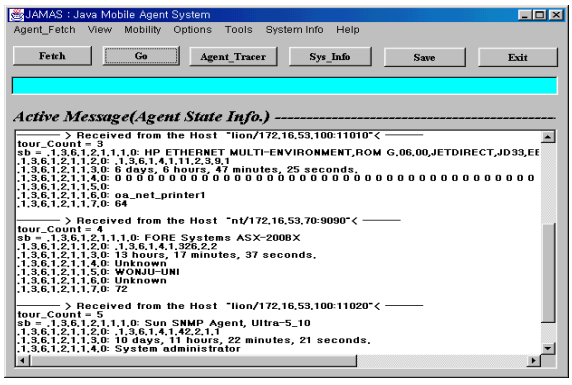


그림 6. 이동에이전트 결과 정보

각 단계에 의해서 이동에이전트가 이주되어 수행된 결과 화면은 그림 6과 같다. 그림 6은 통신망에 연동된 NE들의 MIB 시스템 정보를 접근하여 관리자 GUI에 브라우징 결과를 보여준다. 여기서 이동에이전트는 각 시스템으로 이동되어 실행한 후, 상태 정보를 관리자 창으로 전달하는 호스트 이름과 IP 주소를 매번 나타내고, 또한 해당 시스템 내에서의 멤버 변수에 대한 정보도 지속적으로 전달하였다. 즉, 실험 대상이 된 각 NE마다 MIB의 객체 식별자에 대한 정보가 구조체 형식의 객체 식별자를 갖고서 관리자의 창에 전달되었다.

또한, JAMAS의 사용자 인터페이스는 부가적 기능으로

서 이동에이전트가 이동하는 동안 실행 정보로서 어디에서 어떻게 동작되고 있는지, 처리 시간은 얼마나 되는지 등 에이전트 수행과 관련된 정보를 제공하는 에이전트 추적 GUI를 지원한다. 이러한 기능을 제공하는 그림 7 Agent_Tracer 화면은 이동에이전트가 출발하여 각 목적지 노드에서 종료된 시간과 결과를 되돌려 주는데 걸리는 시간, 즉 이동에이전트의 이주 시작부터 최종 목적지에서의 작업 완료까지인 생명 주기 시간을 밀리초(millisecond)로 알려준다.

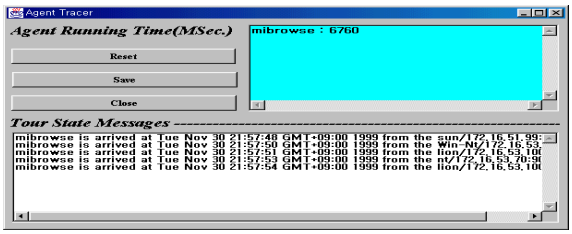


그림 7. Agent_Tracer 화면

3.2 구현 2 모델

그룹 협력 작업에 의한 망 관리 구현 모델은 JAMAS에서 제공하는 간접 미팅 메커니즘과 효율적인 정보 저장소인 Info_Pool을 이용하여 구현한다[11]. 이를 위해서, 4개의 각 NE들에 대해 3개의 작업에이전트와 1개의 주에이전트간의 메시지 전달 수행이 이루어져 협력 작업을 올바르게 수행하는 것으로서 그림 8처럼 수행한다. 여기서 N개의 NE를 가정하였을 때, N-1개의 작업에이전트만을 이용하는 이유는 주에이전트도 단순히 정보 수집하는 기능을 수행하는 것이 아니라 작업에이전트처럼 동일한 작업과 메시지 수집을 동시에 수행함을 의미하기 때문이다.

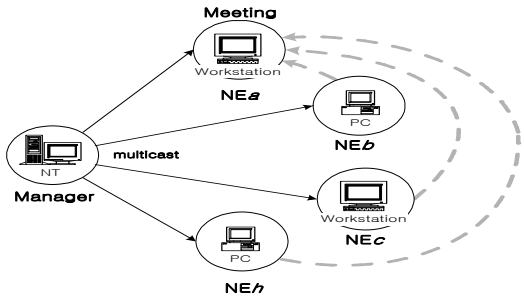


그림 8. 그룹 협력 작업 사례

그러므로, 그림 8처럼 그룹 협력 작업 수행은 4개의 에이전트가 관리자에 의해서 멀티캐스트되었을 때, 주에이전트 'master'는 NE*a*에, 나머지 작업에이전트들(여기서 'worker_1', 'worker_2', 'worker_3'이라 한다)은 각각 NE*b*, NE*c*, NE*h*로 동시에 이동된다. 이후, 각 작업에이전트들의 결과는 주에이전트가 수집하기 위해 이동될 수도 있으나, 미팅 장소로의 메시지 전달 기능이 있으므로 주에이전트가 있는 미팅 장소인 NE*a*의 Info_Pool에 메시지 전달을 수행한다. 결국 주에이전트는 자기의 실행 결과와 각 작업에이전트들이 보내온 결과를 수집하여 관리자에게 최종 전달한다.

이를 위해서 그룹 협력 작업에 참여하기 위한 주/작업에이전트 모두를 그림 9처럼 적재한 후, 'Go' 버튼에 의해 멀티캐스트를 수행한다. 이후 멀티캐스트(multicast)된 각 에이전트들은 1회씩만 방문하고 종료된다. 그리고, 각 에이전트들이 각 NE의 Info_Pool에 전달한 메시지들을 JAMAS는 즉각적으로 미팅 장소인 NE*a*의 Info_Pool에 메시지 전달을 수행한다.

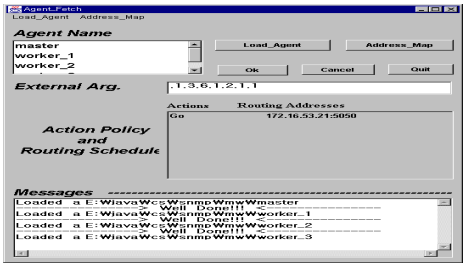
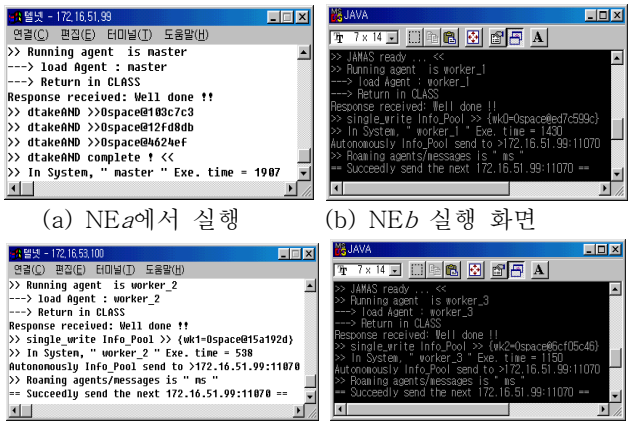


그림 9. 그룹 협력을 위한 이동에이전트들

이러한 일련의 실행 과정을 나타낸 지역 NE들의 캡쳐 화면은 그림 10과 같다. 그림 10 (b),(c),(d)는 각 작업에이전트들은 수행 결과를 지역 NE의 Info_Pool에 객체형의 메시지인 Opsace를 기록하였으며, 동시에 각 NE에 탑재되어 있는 JAMAS가 그림 10(a)인 NE*a*의 주소로 메시지 전달을 한다. 즉, 메시지 전달을 이동에이전트 자신에서 하는 것이 아니라 JAMAS에서 지원하므로 에이전트 자체의 역할이 감소하게 된다. 이후, 그림 10(a)처럼 주에이전트는 자신의 지역 MIB를 먼저 가져오고, JAMAS의 회의 메소드를 이용하여 각 NE*b*, NE*c*, NE*h*에서 보내어진 메시지를 한꺼번에 읽어들인다.



(a) NE*a*에서 실행 (b) NE*b* 실행 화면

(c) NE*c* 실행 화면 (d) NE*h* 실행 화면

그림 10. 주/작업에이전트들의 실행과 메시지 전달

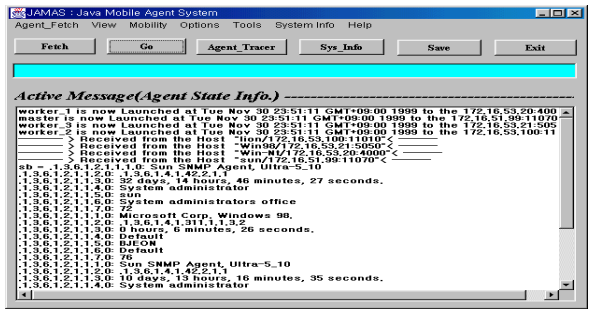


그림 11. 이동에이전트들의 결과 정보

관리자의 창으로서 그림 11은 4개의 주/작업에이전트들이 동시에 이동을 시작하였음을 나타내며, 아울러 결손이 없는 주/작업에이전트들의 도착 정보를 보여준다. 여기서 나타난 상태 정보는 주에이전트에 대한 상태 정보만 보여진 것이다. 그러나 주에이전트는 각 작업에이전트가 전달한 MIB 시스템 정보뿐만 아니라 자신의 지역 MIB 정보

에 대한 접근 결과도 관리자에게 브라우징하였다. 그림 11의 결과에 의하면 각 NE들에 대한 MIB 정보를 가져오는 그룹 협력 작업의 주/작업에이전트들을 이용하였더라도 3.1절의 구현처럼 동일한 결과이다. 또, 주에이전트는 각 작업에이전트들이 수행한 결과를 수집하기 위해 다른 NE들로 이동할 필요가 없이 자신의 미팅 장소에서 메시지를 수집하면 되었고, 또한 작업에이전트도 자신의 결과를 개별적으로 관리자에게 전달하지 않았다. 따라서 종래의 클라이언트/서버 시스템에 의한 MIB 브라우저의 병목 현상을 방지할 수 있다.

4. 성능 평가

망 관리를 위해 사용된 이동에이전트와 JAMAS의 오버헤드 비용을 산출하여 이에 대한 평가를 분석한다. 이를 위하여 기존의 클라이언트/서버 방식에 의해서 4개의 지역 NE들에서 MIB 정보를 가져오는 기존 모델과 3.1절의 구현 1 모델, 그리고 그룹 협력 작업의 구현 3 모델을 가정하였을 때, 이들에 대한 처리비용을 비교 평가한다. 이를 위해서 실험에 사용된 4개의 NE*a*, NE*b*, NE*c*, NE*h*를 대상으로 동일 시간대에 각 모델을 10회씩 수행하여, 이를 평가 및 분석한다.

표 2. 각 NE에서의 응용 모델간의 지역 실행비용
(단위 : millisecond)

각 NE 실행 / 실험 회수	NE*a*			NE*b*			NE*c*			NE*h*			총비용		
	기존 모델	구현 1 모델	구현 2 모델	기존 모델	구현 1 모델	구현 2 모델	기존 모델	구현 1 모델	구현 2 모델	기존 모델	구현 1 모델	구현 2 모델	기존 모델	구현 1 모델	구현 2 모델
1	569	535	2091	1210	1210	1480	571	538	537	940	1040	1050	3290	5580	2580
2	575	525	1927	1150	1210	1320	564	569	545	990	1100	1100	3279	5600	2420
3	572	527	1860	1270	1210	1320	616	557	540	1050	990	1040	3508	5610	2310
4	576	536	1902	1210	1210	1430	564	534	537	940	1050	1100	3290	5490	2300
5	651	533	1870	1200	1210	1320	565	546	539	990	1040	1100	3406	5600	2310
6	573	528	2015	1210	1200	1430	564	517	538	1100	990	1040	3447	5490	2420
7	573	528	2175	1210	1210	1370	565	536	636	930	1040	1100	3278	5550	2630
8	573	529	2121	1200	1210	1430	565	537	540	1040	1040	1260	3378	5540	2580
9	572	528	2070	1210	1210	1430	660	534	540	990	990	1160	3432	5490	2530
10	573	529	2028	1210	1210	1380	564	536	574	930	1040	1100	3277	5550	2410
평균	581	530	2006	1208	1209	1391	580	540	553	990	1032	1105	3359	5550	2449
표준 편차	25	4	111	29	3	57	32	14	31	58	34	66	86	48	124
최대 편차	82	11	315	120	10	160	96	52	99	170	110	120	231	120	330

첫 번째로, 이동에이전트와 시스템 자체에 대한 고유 비용을 평가하기 위해 기존 모델과 구현 1 모델에 대해서 평가한다. 즉, 구현 시나리오 1을 적용한 이동에이전트 'mibrowse'가 4개의 NE들인 NE*a* → NE*b* → NE*c* → NE*h* 순으로 이주 및 실행한 비용과 기존 모델로서 4개의 NE들에서 실행한 비용을 서로 분석하였다. 이에 대한 실험 결과는 표 2와 같으며, 여기서 총비용이라 함은 표 2의 기존 모델을 제외하였을 때, 통신비용을 포함한 것까지를 의미한다. 이 표에 따르면 각 지역 NE내에서는 평균 처리비용은 구현 1 모델이 기존 모델에 비해서 NE*h*를 제외하고는 대체로 같거나 적기 때문에 효율적이다. 아울러 구현 1 모델에 대한 에이전트의 지역 실행비용도 기존 모델에 대한 것보다 편차가 매우 적어 신뢰성있게 실행되고 있음을 나타낸다. 다만, JAMAS에서는 이동에이전트가 각 NE에서 수행하였던 정보를 관리자에게 매번 전달하기 때문에 이에 대한 통신비용이 표 2처럼 구현 1 모델의 통신비용이 JAMAS의 오버헤드 비용이다. 또, 표 2의 총비용은 구현 1 모델이 기존 모델보다 더 늦지만, 이는 기존 모델에서 통신비용을 무시한 것이기 때문이며, 표 2에서 구현 1 모델의 평균 통신비용을 동일하게 적용하였을 경우 기존 모델은 5.598초가 되며, 이는 JAMAS를 이용한 구현 1 모델이 다소 효율적임을 증명한다.

두 번째로, 그룹 협력 작업의 효율성 평가를 위해 기존 모델과 구현 1과 2 모델에 대해서 상호 비교 분석한다.

그림 12에 나타난 바와 같이 각 모델을 수행하는 시스템 NE들의 실행 편차는 시간상 거의 무시할 정도로 비슷하다. 그러나, 그림 12의 각 NE들의 평균 지역 실행비용은 구현 2 모델이 대부분 크다. 그 이유는 작업에이전트들이 실행하고 미팅 장소로 메시지를 전달하여야 하는 비용을 갖고 있기 때문이다. 특히, 주에이전트가 작업에이전트들로부터 메시지를 수집하여야 하는 동기화 비용 때문에 NE*a*에서 구현 2 모델은 다른 모델과 비교해서 2배 이상의 지역 실행비용이 소요됐다.

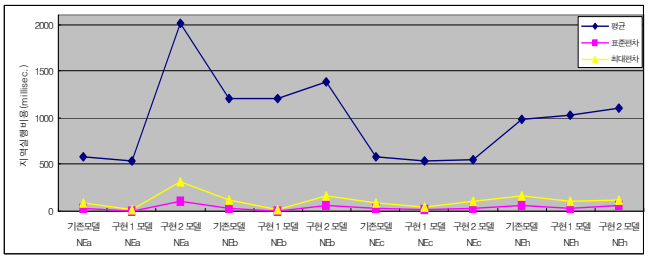


그림 12. 각 모델별 지역 실행비용의 평균과 표준 편차, 최대 편차

구현 1 모델과 구현 2 모델간의 비교에서 관리자에게로의 결과 정보 전달을 위한 통신비용은 표 3과 같다. 표 3과 그림 13처럼 대체로 구현 2 모델이 구현 1 모델보다 평균 통신비용은 작다. 그러나 두 모델간 통신비용에 대한 편차 폭이 다소 크며, 특히 표 2의 편차와 비교하였을 때도 매우 크다. 그 이유는 실험 환경이 불규칙적인 통신망 트래픽으로 인해 다소 불안정한 통신망 오버헤드 요소로 간주된다. 이같은 차이에도 불구하고, 표 3의 NE*a*에서는 구현 1과 2 모델간의 통신비용에 있어 구현 2 모델이 보다 더 크다. 왜냐하면, 수집된 메시지 양이 구현 1 모델의 메시지 양보다 더 크기 때문에 이에 대한 전송 비용이 반영되었음을 알 수 있다.

표 3. 각 NE에서의 응용 모델간의 통신비용
(단위 : millisecond)

통신 비용 / 실험 수	NE*a*		NE*b*		NE*c*		NE*h*		총비용	
	구현 1 모델	구현 2 모델	구현 1 모델	구현 2 모델	구현 1 모델	구현 2 모델	구현 1 모델	구현 2 모델	구현 1 모델	구현 2 모델
1	455	489	760	1050	622	613	420	270	2257	2422
2	405	493	770	760	641	495	380	160	2196	1908
3	413	450	820	660	593	510	500	230	2326	1850
4	394	398	820	660	626	503	320	110	2160	1671
5	397	440	830	710	604	511	440	110	2271	1771
6	402	405	830	770	533	562	490	270	2255	2007
7	462	455	820	710	504	344	450	160	2236	1669
8	461	459	710	880	613	500	440	340	2224	2179
9	402	460	770	820	616	450	440	210	2228	1940
10	401	382	770	820	674	466	390	210	2235	1878
평 균	419	443	790	784	603	495	427	207	2239	1930
표준편차	28	37	40	118	50	70	53	74	44	231
최대편차	66	111	120	390	170	269	180	230	166	753

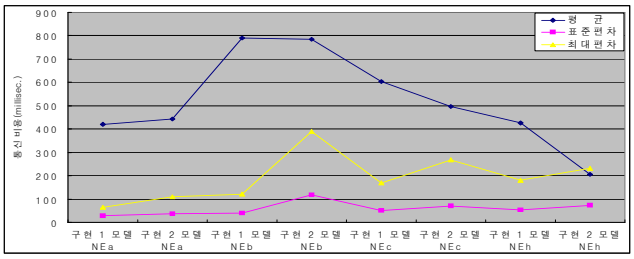


그림 13. 두 모델의 통신비용에 대한 평균과 표준 편차, 최대 편차

이론적으로 동일 조건하에서 구현 1 모델과 구현 2 모델간의 성능 평가는 구현 2 모델이 구현 1 모델보다 4배 정도 높아야 한다. 왜냐하면 구현 1 모델은 1개의 이동에이전트가 4개의 NE들을 모두 방문하지만 구현 2 모델은 4개의 이동에이전트가 1번씩만 방문하기 때문이다. 그러나 실제 실험에서는 표 2와 다음 그림 14에서 나타났듯 성능 개선은 평균적으로 약 2.3배정도 밖에 개선되지 않았다. 이에 대한 주된 이유는 그룹 협력 작업을 하는 각 작업에이전트의 실행 속도 및 통신망을 통한 메시지 전달 시간과 주에이전트에서 메시지 수집을 위한 동기화 비용 때문이다. 즉, 4개의 에이전트가 동시에 실행된다 할 지라도 모든 상태 정보 전달이 관리자에게 전달되는 통신비용과 동시에 주에이전트는 모든 작업에이전트가 전달한 메시지를 수집해야 하는 동기화 비용이 필요하기 때문에 기대 이상의 성능 향상은 되지 않았다.

그림 14는 성능 평가를 한 응용 모델들의 실행 총비용을 나타낸 것이다. 그리고, 통신비용이 없는 기존 모델에 비해 구현 2 모델은 평균적으로 약 1.4배정도 개선된 성능 향상을 나타냈으며, 전술한 바와 같이 기존 모델에 통신비용을 적용하였을 경우 구현 1 모델처럼 약 2.3배 정도 성능을 개선한다. 이같은 분석 결과를 바탕으로 가정하였을 때, 관리해야 할 NE들의 수가 많으면 많을수록 더 나은 성능 향상을 기대할 수 있다.

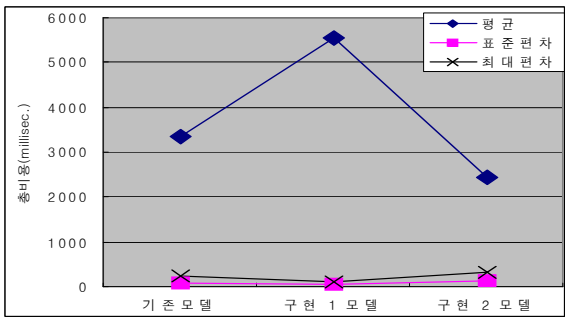


그림 14. 응용 모델들의 실행 총비용

5. 결론

본 논문에서 제안된 JAMAS 기반 이동에이전트는 SNMP 에이전트의 변형없이 지역/원격지의 NE들과 대화를 통해 MIB 정보 접근 가능함을 보이는 브라우저 역할을 수행하였다. 특히, 2개의 구현 모델에 의한 성능 분석 결과에 의하면, 기존 통신망 관리 방식인 클라이언트/서버 모델에 의한 관리자/SNMP 에이전트들간의 원격 대화로 인한 비효율과 병목 문제를 제안된 이동에이전트를 이용할 경우 효율적으로 개선될 수 있다. 특히, 그룹에이전트 모델을 적용한 그룹 협력 작업은 보다 효율적인 성능 향상이 되었다. 게다가 실행 환경적인 측면을 고려할 경우, 이형의 NE들에 자바 가상 머신과 이동에이전트 시스템이 탑재될 경우 효율적인 통합 관리 환경이 된다.

향후 연구 과제는 JAMAS 사용자 인터페이스를 NMS용으로 변환이 필요하며, CMIP도 통합 지원할 수 있는 응용 이동에이전트가 요구된다. 또한, 이동에이전트 컴포넌트에 지능을 지원하면 망 관리 전문가 시스템을 구현할 수 있기 때문에 이에 대한 연구 개발도 필요하다.

참고문헌

[1] Advent Network Management, Inc., http://www.adventnet.com

[2] A. Bieszczad, B. Pagurek, T. White, “Mobile Agents for Network Management”, IEEE Communications Surveys, Sept. 1998.

[3] D. Lange, M. Oshima, “Seven good reasons for mobile agents”, CACM, Vol. 42(3), Mar. 1999, pp 88-89.

[4] G. Luderer et. al,"Network Management Agents Supported by A Java Environment" TR, Arizona St. Univ., 1996.

[5] B. Pagurek, Y. Wang, T. White, “Integration of Mobile Agents with SNMP: How and Why. Submitted to NOMS '2000.

[6] A. Puliafito et al., “A Java-based Distributed Network Management Architecture”, 3rd Int'l Conf. on Computer Science and Informatics (CS&I'97), Mar. 1997.

[7] D. W. Stevenson, “Network Management: What it is and what it isn't”, http://www.sce.carleton.ca/netmanage/NetMngmnt/NetMngmnt.html

[8] Sun Microsystems, http://java.sun.com

[9] T. White et al, “Network Modeling For Management Applications Using Intelligent Mobile Agents”, Network and Systems Management, Sept. 1999.

[10] 전병국, 이근상, 최영근, “Java언어를 이용한 객체이동시스템의 설계 및 구현”, 한국정보처리학회 논문지, 6권 1호, Jan. 1999, pp 49-56.

[11] 전병국, “자바 환경 기반의 효율적인 이동에이전트 시스템”, 광운대 박사학위 논문, 1999.